morphing_project_commented

April 25, 2020

1 Image Morphing Project

1.0.1 Submitted to Dr. Anukriti Bansal

1.0.2 By - Rivanshu Goyal (17UCS132) and Daksh Balyan (17UCS049)

[1]:
```
#Importing the Required Libraries
import cv2
import numpy as np
```

1.1 Different Functions defined to be used later in the code

[2]:
```
#Includes the image corners as control points
def get_corner_points(img):
    width = img.shape[0]
    height = img.shape[1]
    corner_points=[]
    corner_points.append((0,0))
    corner_points.append((width,0))
    corner_points.append((0,height))
    corner_points.append((width,height))
    return corner_points

#Mouse Clicks Call Back Functions for both the images
def mouse_click_clinton(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(clinton_img, (x,y), 3, (255,0,0), -1)
    if event == cv2.EVENT_LBUTTONUP:
        control_points_clinton.append((y,x))
    cv2.imshow("Clinton", clinton_img)

def mouse_click_bush(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(bush_img, (x,y), 3, (0,0,255), -1)
    if event == cv2.EVENT_LBUTTONUP:
        control_points_bush.append((y,x))
    cv2.imshow("Bush",bush_img)
```

[3]:
```
#Function to solve linear equation in two variables
def solve_linalg(coeff,const):
    a1 = coeff[0][0]+0.0001
    b1 = coeff[0][1]
    a2 = coeff[1][0]+0.0001          #.......a1*x + b1*y = c1......#
    b2 = coeff[1][1]                 #.......a2*x + b2*y = c2......#
    c1 = const[0]
    c2 = const[1]

    x = (c2*b1 - c1*b2)/(a2*b1-a1*b2)
    y = (c1*a2-c2*a1)/(a2*b1-a1*b2)

    return (x,y)

#Function to get the centre and radius of the circumcircle of a triangle
def circumcircle(trngl):
    x1 = trngl[0][0]
    y1 = trngl[0][1]
    x2 = trngl[1][0]                 #...center of the circle cicumsribed over
    y2 = trngl[1][1]                 #...any triangle is the intersection point
    x3 = trngl[2][0]                 #...of the perpendicular bisectors of the
    y3 = trngl[2][1]                 #...sides of the triangle.

    coeff = np.array([[2*(x1-x2),2*(y1-y2)], [2*(x1-x3),2*(y1-y3)]],dtype=np.
 ↪float64)
    constants = np.
 ↪array([x1**2+y1**2-x2**2-y2**2,x1**2+y1**2-x3**2-y3**2],dtype=np.float64)
    center = solve_linalg(coeff,constants)
    radius_squared = (center[0]-x1)**2 + (center[1]-y1)**2  #...radius is the␣
 ↪euclidean distance between
    return [center,radius_squared]                          #...center and one␣
 ↪of the vertices of the triangle.

#Function to check if a point lies in the circle
def is_in_circle(centre,radius_sq,point):
    x0 = centre[0]
    y0 = centre[1]                      #...checking if the value after putting
    x  = point[0]                       #...the point in the equation of the␣
 ↪circle
    y  = point[1]                       #...is negative or equal to zero
    value = (x-x0)**2 + (y-y0)**2 -radius_sq
    if(value<=0):
        return True
    else:
        return False

#Function to reverse the tuple
```

```
def reverse(tup):
    new_tup = tup[::-1]
    return new_tup

#Function to form triangle for given control points
def delaunay(points):
    #...Dictionary for triangles where key is the tuple of vertices
    #...and value is the tuple:(centre,radius_squared)
    triangles = {}
    #Creating a super Triangle
    super_triangle  = ((0,10000),(-10000,-10000),(10000,-10000))
    #Adding it to the current list of triangles
    triangles[super_triangle] = circumcircle(super_triangle)
    #Iterating over each point in the list
    for i in points:
        edges = []              #for the star shaped polygon
        wrong_triangles = []      #for storing the wrong triangles
        #Checking the triangles which violate Delaunay traingulation by the␣
 ↪addition of current point
        for t in triangles:
            centre = triangles[t][0]
            radius = triangles[t][1]
            #Checking if the point lies in the circumcircle of triangle
            if(is_in_circle(centre,radius,i)):
                #adding the wrong triangles to the list
                wrong_triangles.append(t)
                #Calculating the edges of the polygon formed by the wrong␣
 ↪triangles
                t_edge = [(t[0],t[1]),(t[1],t[2]),(t[2],t[0])]
                for e in t_edge:
                    if e in edges:
                        edges.remove(e)
                        continue
                    rev = reverse(e)
                    if rev in edges:
                        edges.remove(rev)
                        continue
                    edges.append(e)
        #removing the wrong triangles from the list
        for t in wrong_triangles:
            del triangles[t]
        #...adding the new triangles formed by the edges of the star shaped␣
 ↪polygon
        #...and the new point
        for e in edges:
            t = (e[0],e[1],i)
            triangles[t] = circumcircle(t)
```

```
    final_triangles = []
    #removing the triangles formed by the vertices of the super triangle
    for t in triangles:
        if super_triangle[0] in t or super_triangle[1] in t or super_triangle[2]␣
 ↪in t:
            continue
        else:
            final_triangles.append(t)
    #returning the final list of triangles as a list of tuples
    return final_triangles

#Function to draw triangles over the given image
def triangle(img, trngle_coord,clr):
    for i in range(len(trngle_coord)):
        cv2.line(img, reverse(trngle_coord[i][0]),reverse(trngle_coord[i][1]),␣
 ↪clr, 2)
        cv2.line(img, reverse(trngle_coord[i][1]),reverse(trngle_coord[i][2]),␣
 ↪clr, 2)
        cv2.line(img, reverse(trngle_coord[i][0]),reverse(trngle_coord[i][2]),␣
 ↪clr, 2)
```

[4]:
```
#Function to find area of the triangle
def area_triangle(x1,y1,x2,y2,x3,y3):
    return abs((x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2))/2)

#Function to check if a point lies inside a triangle
def is_in_triangle(triangle_coord, x, y):
    x1=triangle_coord[0][0]                          #If the point lies inside␣
 ↪the triangle then,
    y1=triangle_coord[0][1]                          #....the sum of the areas of␣
 ↪the triangles
    x2=triangle_coord[1][0]                          #....formed between the␣
 ↪edges of the current
    y2=triangle_coord[1][1]                          #....triangle and the point␣
 ↪equals the area
    x3=triangle_coord[2][0]                          #....of the current triangle
    y3=triangle_coord[2][1]
    A = area_triangle(x1, y1, x2, y2, x3, y3)
    A1 = area_triangle(x, y, x2, y2, x3, y3)
    A2 = area_triangle(x1, y1, x, y, x3, y3)
    A3 = area_triangle(x1, y1, x2, y2, x, y)
    if(A == (A1 + A2 + A3)):
        return True
    else:
        return False
```

1.2 Main Program Starts

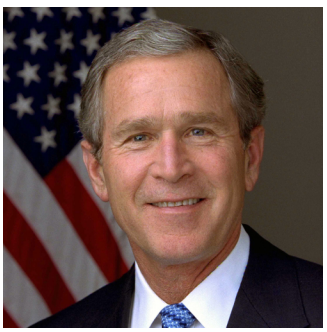
[5]:
```
#Loading the Images
clinton_img = cv2.imread("Clinton.jpg")
bush_img = cv2.imread("Bush.jpg")

#getting the corner points of the image as control points
control_points_clinton = get_corner_points(clinton_img)
control_points_bush = get_corner_points(bush_img)
```

[6]:
```
#Taking control points as input from user
cv2.imshow("Clinton", clinton_img)
cv2.imshow("Bush",bush_img)
cv2.setMouseCallback("Clinton",mouse_click_clinton)
cv2.setMouseCallback("Bush",mouse_click_bush)
cv2.waitKey(0)
cv2.destroyAllWindows()

#Saving the images depicting control points given by user
cv2.imwrite("bush_c.jpg",bush_img)
cv2.imwrite("clinton_c.jpg",clinton_img)
```

[6]:
```
True
```

1.2.1 First Image (Clinton.jpg)

1.2.2 Target/ Final Image (Bush.jpg)



1.2.3 User chosen control points for Clinton.jpg

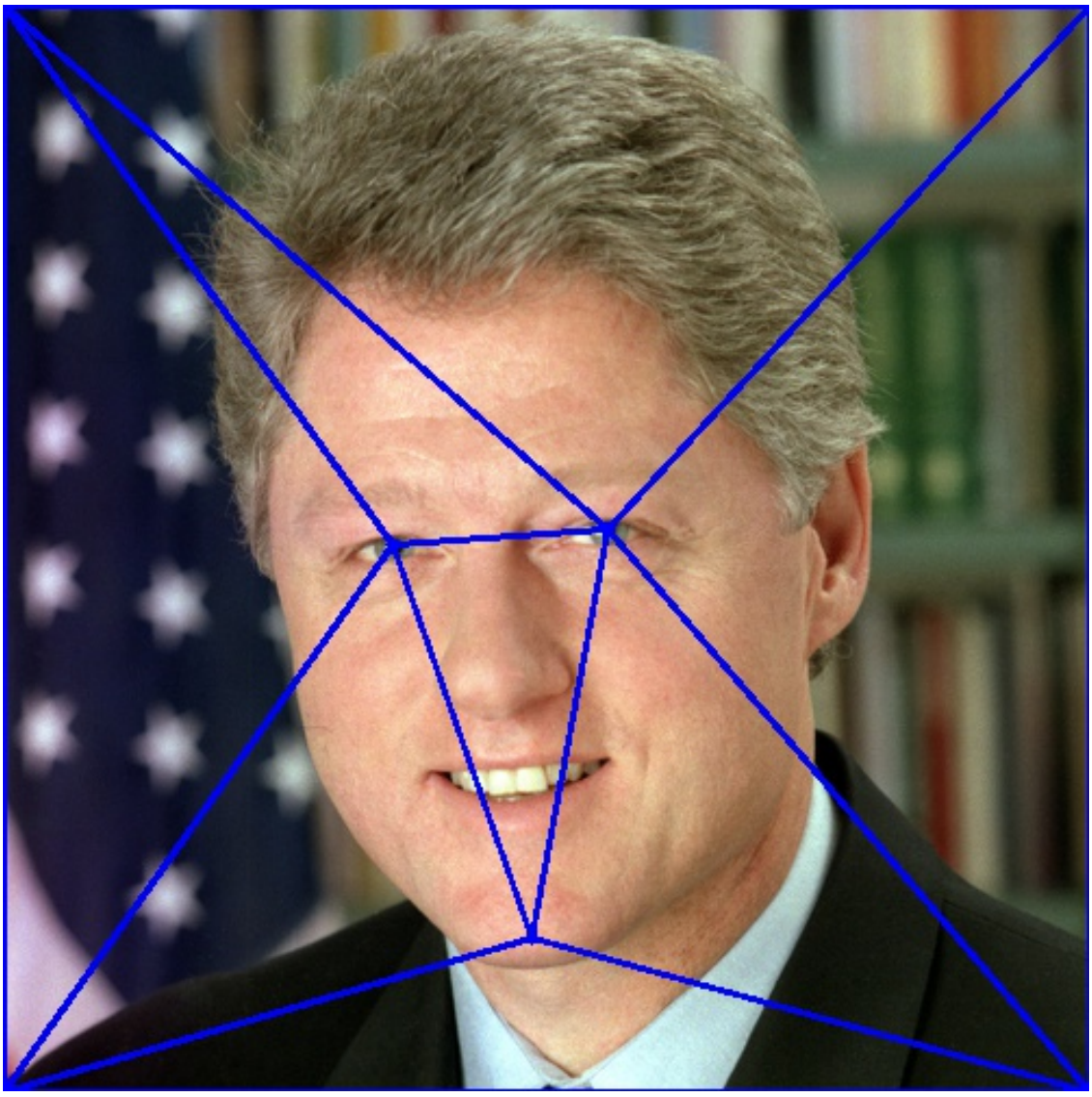
1.2.4 User chosen control points for Bush.jpg

[7]:
```
#Triangulating the Control points using Delaunay Triangulation
triangle_coord_clinton = delaunay(control_points_clinton)
triangle_coord_bush = []
#Finding the triangles in the final image corresponding those of the inital image
for t in triangle_coord_clinton:
    e0 = control_points_bush[control_points_clinton.index(t[0])]
    e1 = control_points_bush[control_points_clinton.index(t[1])]
    e2 = control_points_bush[control_points_clinton.index(t[2])]
    triangle_coord_bush.append((e0,e1,e2))
```

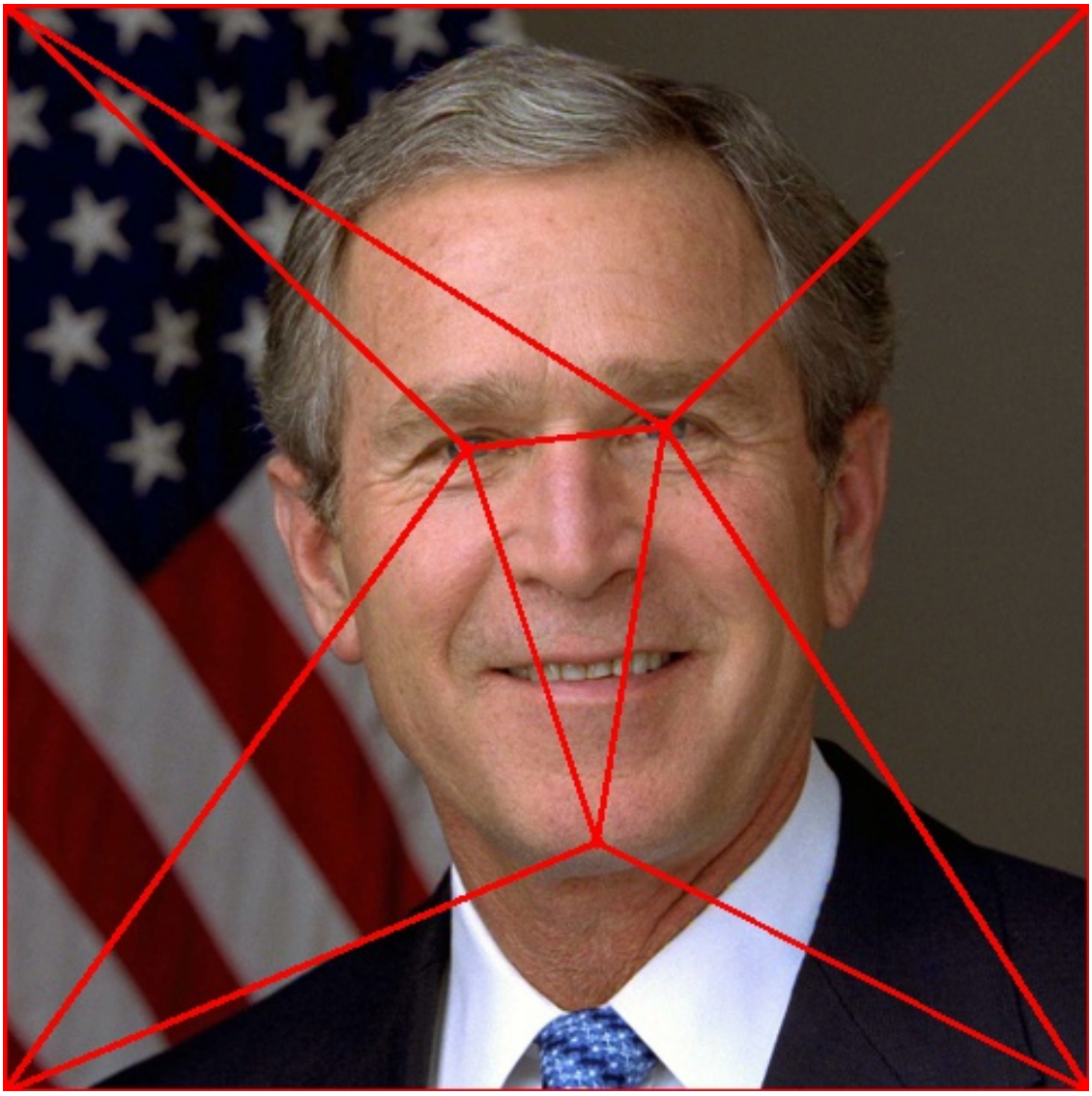
[8]:
```
#Printing the Triangles over the Image
clr_lineClinton = (255,0,0)  #BLUE
clr_lineBush = (0,0,255)     #RED
triangle(clinton_img,triangle_coord_clinton, clr_lineClinton)
triangle(bush_img,triangle_coord_bush, clr_lineBush)

cv2.imshow("clinton",clinton_img)
cv2.imshow("bush",bush_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

#Saving the triangulated Images
cv2.imwrite("clint_tr.jpg",clinton_img)
```

```
cv2.imwrite("bush_tr.jpg",bush_img)
```

1.2.5 Delaunay Triangulated Clinton.jpg



1.2.6 Delaunay Triangulated Bush.jpg



[9]:
```
#Number of frames
no_of_frames = 100
#Array of all the frames
frames = np.array([np.zeros_like(clinton_img)]*no_of_frames)
#Setting the inital and final frame to Initial and Final Image respectively
frames[0] = cv2.imread("Clinton.jpg")
frames[no_of_frames-1] = cv2.imread("Bush.jpg")
```

[10]:
```
#List to store triangle vertices of the frames
frames_triangle_coord = []
#List to store affine basis of each frame of each triangle
frames_affine_basis = []

for k in range(len(frames)):
```

```
    #List to store triangle coordinates of the current frame
    frame_triangle_coord = []
    #List to store affine basis of the triangles of current frame
    frame_affine_basis = []
    for trngl,trngln in zip(triangle_coord_clinton,triangle_coord_bush):
            p0_x = trngl[0][0]
            p0_y = trngl[0][1]
            p1_x = trngl[1][0]
            p1_y = trngl[1][1]
            p2_x = trngl[2][0]
            p2_y = trngl[2][1]

            p0_xn = trngln[0][0]
            p0_yn = trngln[0][1]
            p1_xn = trngln[1][0]
            p1_yn = trngln[1][1]
            p2_xn = trngln[2][0]
            p2_yn = trngln[2][1]

            #Linearly Interpolating the control points to find the corresponding␣
↪control points in intermediate frames
            p0_xk = int(((no_of_frames-k)/no_of_frames)*p0_x + (k/
↪no_of_frames)*p0_xn)
            p0_yk = int(((no_of_frames-k)/no_of_frames)*p0_y + (k/
↪no_of_frames)*p0_yn)
            p1_xk = int(((no_of_frames-k)/no_of_frames)*p1_x + (k/
↪no_of_frames)*p1_xn)
            p1_yk = int(((no_of_frames-k)/no_of_frames)*p1_y + (k/
↪no_of_frames)*p1_yn)
            p2_xk = int(((no_of_frames-k)/no_of_frames)*p2_x + (k/
↪no_of_frames)*p2_xn)
            p2_yk = int(((no_of_frames-k)/no_of_frames)*p2_y + (k/
↪no_of_frames)*p2_yn)

            #Finding the affine basis of the current triangle
            e1_xk = p1_xk - p0_xk
            e1_yk = p1_yk - p0_yk
            e2_xk = p2_xk - p0_xk
            e2_yk = p2_yk - p0_yk


            frame_affine_basis.append(((e1_xk,e1_yk),(e2_xk,e2_yk)))
            frame_triangle_coord.
↪append(((p0_xk,p0_yk),(p1_xk,p1_yk),(p2_xk,p2_yk)))
    frames_triangle_coord.append(frame_triangle_coord)
    frames_affine_basis.append(frame_affine_basis)
```

[11]:
```
#Traversing over each frame
for k in range(1,len(frames)-1):
    #Traversing over each pixel of the frame
    for x in range(frames[k].shape[0]):
        for y in range(frames[k].shape[1]):
            count=0
            #Finding which triangle the pixel belongs to in the current pixel
            for triangle_coord,affine_basis in␣
↪zip(frames_triangle_coord[k-1],frames_affine_basis[k-1]):
                if(is_in_triangle(triangle_coord,x,y)):
                    #Calculating alpha and beta i.e. affine coordinates
                    coeff = np.
↪array([[affine_basis[0][0],affine_basis[1][0]],[affine_basis[0][1],affine_basis[1][1]]])
                    constants = np.
↪array([x-triangle_coord[0][0],y-triangle_coord[0][1]])
                    z = solve_linalg(coeff, constants)
                    alpha = z[0]
                    beta = z[1]

                    #Calculating corresponding points in source and destination␣
↪image
                    p_x = alpha*frames_affine_basis[0][count][0][0] +␣
↪beta*frames_affine_basis[0][count][1][0] +␣
↪frames_triangle_coord[0][count][0][0]
                    p_y = alpha*frames_affine_basis[0][count][0][1] +␣
↪beta*frames_affine_basis[0][count][1][1] +␣
↪frames_triangle_coord[0][count][0][1]
                    p_xn =␣
↪alpha*frames_affine_basis[no_of_frames-1][count][0][0] +␣
↪beta*frames_affine_basis[no_of_frames-1][count][1][0] +␣
↪frames_triangle_coord[no_of_frames-1][count][0][0]
                    p_yn =␣
↪alpha*frames_affine_basis[no_of_frames-1][count][0][1] +␣
↪beta*frames_affine_basis[no_of_frames-1][count][1][1] +␣
↪frames_triangle_coord[no_of_frames-1][count][0][1]

                    #Adjusting the values
                    if(p_x>=frames[k].shape[0]):
                        p_x = frames[k].shape[0]-1
                    if(p_y>=frames[k].shape[1]):
                        p_y = frames[k].shape[1]-1
                    if(p_xn>=frames[k].shape[0]):
                        p_xn = frames[k].shape[0]-1
                    if(p_yn>frames[k].shape[1]):
                        p_yn = frames[k].shape[1]-1
```

```
                    #Assigning the pixel intensity
                    pixel_colour = ((1-k/
↪no_of_frames)*frames[0][int(p_x),int(p_y)] + (k/
↪no_of_frames)*frames[no_of_frames-1][int(p_xn),int(p_yn)]).astype(int)
                    frames[k][x][y] = pixel_colour
                    break
                else:
                    count+=1
```

[12]:
```
#Saving the frames
for i in range(no_of_frames):
    name = "img" + str(i+1) + ".jpg"
    cv2.imwrite(name,frames[i])
```