**COMPUTER VISION ASSIGNMENT – 3**

**Submitted by:** Guttikonda NagaTeja (1104370)

**FLOW DIAGRAM:**

RUN THE PROGRAM

Morphed image from two pics

Generate the feature points for two images

Morph Images

Detect Feature pts

OPEN TWO Images

Delay Triangul 2

Delay Triangul 1

Generate Triangles for face 1

Generate Triangles for face 2

Two images combined o/p

**DESCRIPTION:**

Various functions used in the code are

1)fun\_open\_picture():It will ask to input the data from the computer. It uses the function imread() to read the input file and imshow() to display the output on the user side.

2)fun\_del\_triangulation():It is a function used to generate the triangles on given input images using dlib

3)aff\_transform(): It is used to perform the affine transform using the inbuilt functions warpaffine()

4)morphing triangle():It is used to perform the morphing for given two images using various functions inside such as feautrepoints(),imagine\_facemarks(), where we extract points from the face to perform the efficient morphing.

**CODE:**

**from tkinter import filedialog**

**from collections import OrderedDict**

**import argparse**

**import dlib**

**from tkinter import \***

**import cv2**

**import imutils**

**import numpy as np**

**import sys**

**root = Tk()**

**root.title("Comp Vision Assign 3")**

**root.configure(background='red')**

**root.geometry('300x400')**

**def fun\_open\_picture():**

**inpua = filedialog.askopenfilename()**

**inpub = filedialog.askopenfilename()**

**image1 = cv2.imread(inpua)**

**image2 = cv2.imread(inpub)**

**img\_co=np.hstack((image1,image2))**

**cv2.imshow("image\_opened",img\_co)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**def fun\_del\_triangulation():**

**def rectangle\_has(rect, point) :**

**if point[0] < rect[0] :**

**return False**

**elif point[1] < rect[1] :**

**return False**

**elif point[0] > rect[2] :**

**return False**

**elif point[1] > rect[3] :**

**return False**

**return True**

**# Draw a point**

**def make\_pt(img, p, color ) :**

**img1=cv2.circle( img, p, 2, color, 0 )**

**cv2.imshow(img1,img1)**

**# Draw delaunay triangles**

**def make\_del(img, subdiv, delaunay\_color ) :**

**tridata = subdiv.getTriangleList();**

**size = img.shape**

**r = (0, 0, size[1], size[0])**

**for t in tridata :**

**pa = (t[0], t[1])**

**pb = (t[2], t[3])**

**pt3 = (t[4], t[5])**

**if rectangle\_has(r, pa) and rectangle\_has(r, pb) and rectangle\_has(r, pt3) :**

**cv2.line(img, pa, pb, delaunay\_color,2)**

**cv2.line(img, pb, pt3, delaunay\_color,2)**

**cv2.line(img, pt3, pa, delaunay\_color,2)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**# Define window names**

**win\_delaunay = "Delaunay Triangulation"**

**# Turn on animation while drawing triangles**

**animate = True**

**# Define colors for drawing.**

**delaunay\_color = (0,255,0)**

**points\_color = (0, 0, 255)**

**# Read in the image.**

**img = cv2.imread("dhoni.jpg")**

**#np.savetxt("tri.txt",img,fmt ='%.0f')**

**print(img)**

**# Keep a copy around**

**img\_orig = img.copy();**

**# Rectangle to be used with Subdiv2D**

**size = img.shape**

**rect = (0, 0, size[1], size[0])**

**# Create an instance of Subdiv2D**

**subdiv = cv2.Subdiv2D(rect);**

**# Create an array of points.**

**points = [];**

**# Read in the points from a text file**

**with open('image1.txt') as file :**

**for line in file :**

**x, y = line.split()**

**points.append((int(x), int(y)))**

**# Insert points into subdiv**

**for p in points :**

**subdiv.insert(p)**

**# Show animation**

**if animate :**

**img\_copy = img\_orig.copy()**

**# Draw delaunay triangles**

**make\_del( img\_copy, subdiv, (255, 255, 255) );**

**cv2.imshow(win\_delaunay, img\_copy)**

**cv2.waitKey(100)**

**# Draw delaunay triangles**

**make\_del( img, subdiv, (255, 255, 255) );**

**# Draw points**

**#for p in points :**

**#make\_pt(img, p, (0,0,255))**

**cv2.imshow(win\_delaunay,img)**

**cv2.waitKey(0)**

**def fun\_del\_triangulation2():**

**def rectangle\_has(rect, point) :**

**if point[0] < rect[0] :**

**return False**

**elif point[1] < rect[1] :**

**return False**

**elif point[0] > rect[2] :**

**return False**

**elif point[1] > rect[3] :**

**return False**

**return True**

**# Draw a point**

**def make\_pt(img, p, color ) :**

**img1=cv2.circle( img, p, 2, color, 0 )**

**cv2.imshow(img1,img1)**

**# Draw delaunay triangles**

**def make\_del(img, subdiv, delaunay\_color ) :**

**tridata = subdiv.getTriangleList();**

**size = img.shape**

**r = (0, 0, size[1], size[0])**

**for t in tridata :**

**pa = (t[0], t[1])**

**pb = (t[2], t[3])**

**pt3 = (t[4], t[5])**

**if rectangle\_has(r, pa) and rectangle\_has(r, pb) and rectangle\_has(r, pt3) :**

**cv2.line(img, pa, pb, delaunay\_color,2)**

**cv2.line(img, pb, pt3, delaunay\_color,2)**

**cv2.line(img, pt3, pa, delaunay\_color,2)**

**if \_\_name\_\_ == '\_\_main\_\_':**

**# Define window names**

**win\_delaunay = "Delaunay Triangulation"**

**# Turn on animation while drawing triangles**

**animate = True**

**# Define colors for drawing.**

**delaunay\_color = (0,255,0)**

**points\_color = (0, 0, 255)**

**# Read in the image.**

**img = cv2.imread("teja.jpg")**

**#np.savetxt("tri.txt",img,fmt ='%.0f')**

**print(img)**

**# Keep a copy around**

**img\_orig = img.copy();**

**# Rectangle to be used with Subdiv2D**

**size = img.shape**

**rect = (0, 0, size[1], size[0])**

**# Create an instance of Subdiv2D**

**subdiv = cv2.Subdiv2D(rect);**

**# Create an array of points.**

**points = [];**

**# Read in the points from a text file**

**with open('image2.txt') as file :**

**for line in file :**

**x, y = line.split()**

**points.append((int(x), int(y)))**

**# Insert points into subdiv**

**for p in points :**

**subdiv.insert(p)**

**# Show animation**

**if animate :**

**img\_copy = img\_orig.copy()**

**# Draw delaunay triangles**

**make\_del( img\_copy, subdiv, (255, 255, 255) );**

**cv2.imshow(win\_delaunay, img\_copy)**

**cv2.waitKey(100)**

**# Draw delaunay triangles**

**make\_del( img, subdiv, (255, 255, 255) );**

**# Draw points**

**#for p in points :**

**#make\_pt(img, p, (0,0,255))**

**cv2.imshow(win\_delaunay,img)**

**cv2.waitKey(0)**

**def img\_morphing():**

**def get\_points(path):**

**points = [];**

**with open(path) as file:**

**for line in file:**

**x, y = line.split()**

**points.append((int(x), int(y)))**

**return points**

**def aff\_transform(src, srcTri, dstTri, size):**

**# Finding the affine transform**

**warpMat = cv2.getAffineTransform(np.float32(srcTri), np.float32(dstTri))**

**# Applying affine transform**

**dst = cv2.warpAffine(src, warpMat, (size[0], size[1]), None, flags=cv2.INTER\_LINEAR,**

**borderMode=cv2.BORDER\_REFLECT\_101)**

**return dst**

**def morphing\_Triangle(img1, img2, img, a, b, t, alpha):**

**r1 = cv2.boundingRect(np.float32([a]))**

**r2 = cv2.boundingRect(np.float32([b]))**

**r = cv2.boundingRect(np.float32([t]))**

**Rect\_a = []**

**Rect\_b = []**

**Rect\_t = []**

**for i in range(0, 3):**

**Rect\_t.append(((t[i][0] - r[0]), (t[i][1] - r[1])))**

**Rect\_a.append(((a[i][0] - r1[0]), (a[i][1] - r1[1])))**

**Rect\_b.append(((b[i][0] - r2[0]), (b[i][1] - r2[1])))**

**mask = np.zeros((r[3], r[2], 3), dtype=np.float32)**

**cv2.fillConvexPoly(mask, np.int32(Rect\_t), (1.0, 1.0, 1.0), 16, 0);**

**image1\_Rect = img1[r1[1]:r1[1] + r1[3], r1[0]:r1[0] + r1[2]]**

**image2\_Rect = img2[r2[1]:r2[1] + r2[3], r2[0]:r2[0] + r2[2]]**

**size = (r[2], r[3])**

**change\_img1 = aff\_transform(image1\_Rect, Rect\_a, Rect\_t, size)**

**change\_img2 = aff\_transform(image2\_Rect, Rect\_b, Rect\_t, size)**

**image\_Rect = (1.0 - alpha) \* change\_img1 + alpha \* change\_img2**

**img[r[1]:r[1] + r[3], r[0]:r[0] + r[2]] = img[r[1]:r[1] + r[3], r[0]:r[0] + r[2]] \* (1 - mask) + image\_Rect \* mask**

**if \_\_name\_\_ == '\_\_main\_\_':**

**fname1 = filedialog.askopenfilename()**

**fname2 = filedialog.askopenfilename()**

**alpha = 0.5**

**img1 = cv2.imread(fname1)**

**#filename1=cv2.imwrite(fname1)**

**img2 = cv2.imread(fname2)**

**img1 = np.float32(img1)**

**img2 = np.float32(img2)**

**points1 = get\_points('image1.txt')**

**points2 = get\_points('image2.txt')**

**points = [];**

**for i in range(0, len(points1)):**

**x = (1 - alpha) \* points1[i][0] + alpha \* points2[i][0]**

**y = (1 - alpha) \* points1[i][1] + alpha \* points2[i][1]**

**points.append((x, y))**

**image\_Morph = np.zeros(img1.shape, dtype=img1.dtype)**

**with open("tri.txt") as file:**

**for line in file:**

**x, y, z = line.split()**

**x = int(x)**

**y = int(y)**

**z = int(z)**

**a = [points1[x], points1[y], points1[z]]**

**b = [points2[x], points2[y], points2[z]]**

**t = [points[x], points[y], points[z]]**

**morphing\_Triangle(img1, img2, image\_Morph, a, b, t, alpha)**

**cv2.imshow("Morphed Face", np.uint8(image\_Morph))**

**cv2.waitKey(0)**

**def featurepoints():**

**facial\_features\_cordinates = {}**

**Extended\_points={}**

**Extended\_points=([538,410],**

**[215,727],**

**[0,760],**

**[599,597],**

**[0,0],**

**[0,400],**

**[0,799],**

**[300,799],**

**[599,799],**

**[599,400],**

**[599,0],**

**[300,0])**

**# define a dictionary that maps the indexes of the facial**

**# landmarks to specific face regions**

**FACIAL\_LANDMARKS\_INDEXES = OrderedDict([**

**("Mouth", (48, 68)),**

**("Right\_Eyebrow", (17, 22)),**

**("Left\_Eyebrow", (22, 27)),**

**("Right\_Eye", (36, 42)),**

**("Left\_Eye", (42, 48)),**

**("Nose", (27, 35)),**

**("Jaw", (0, 17))**

**])**

**def shape\_to\_numpy\_array(shape, dtype="int"):**

**# initialize the list of (x, y)-coordinates**

**coordinates = np.zeros((68, 2), dtype=dtype)**

**# loop over the 68 facial landmarks and convert them**

**# to a 2-tuple of (x, y)-coordinates**

**for i in range(0, 68):**

**coordinates[i] = (shape.part(i).x, shape.part(i).y)**

**# return the list of (x, y)-coordinates**

**return coordinates**

**def imagine\_face\_marks(image, shape, colors=None, alpha=0.75):**

**# create two copies of the input image -- one for the**

**# overlay and one for the final output image**

**overlay = image.copy()**

**output = image.copy()**

**# if the colors list is None, initialize it with a unique**

**# color for each facial landmark region**

**if colors is None:**

**colors = [(19, 199, 109), (79, 76, 240), (230, 159, 23),**

**(168, 100, 168), (158, 163, 32),**

**(163, 38, 32), (180, 42, 220)]**

**# loop over the facial landmark regions individually**

**for (i, name) in enumerate(FACIAL\_LANDMARKS\_INDEXES.keys()):**

**# grab the (x, y)-coordinates associated with the**

**# face landmark**

**(j, k) = FACIAL\_LANDMARKS\_INDEXES[name]**

**pts = shape[j:k]**

**facial\_features\_cordinates[name] = pts**

**# check if are supposed to draw the jawline**

**if name == "Jaw":**

**# since the jawline is a non-enclosed facial region,**

**# just draw lines between the (x, y)-coordinates**

**for l in range(1, len(pts)):**

**ptA = tuple(pts[l - 1])**

**ptB = tuple(pts[l])**

**cv2.line(overlay, ptA, ptB, colors[i], 2)**

**# otherwise, compute the convex hull of the facial**

**# landmark coordinates points and display it**

**else:**

**hull = cv2.convexHull(pts)**

**cv2.drawContours(overlay, [hull], -1, colors[i], -1)**

**# apply the transparent overlay**

**cv2.addWeighted(overlay, alpha, output, 1 - alpha, 0, output)**

**# return the output image**

**print(facial\_features\_cordinates)**

**return output**

**# initialize dlib's face detector (HOG-based) and then create**

**# the facial landmark predictor**

**detector = dlib.get\_frontal\_face\_detector()**

**predictor = dlib.shape\_predictor("shape\_predictor\_68\_face\_landmarks.dat")**

**# load the input image, resize it, and convert it to grayscale**

**image = cv2.imread("dhoni.jpg")**

**gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**

**# detect faces in the grayscale image**

**rects = detector(gray, 1)**

**# loop over the face detections**

**for (i, rect) in enumerate(rects):**

**# determine the facial landmarks for the face region, then**

**# convert the landmark (x, y)-coordinates to a NumPy array**

**shape = predictor(gray, rect)**

**shape = shape\_to\_numpy\_array(shape)**

**shapeExt=np.concatenate((shape,Extended\_points))**

**#print(shape)**

**print(shapeExt)**

**np.savetxt("image1.txt",shapeExt,fmt ='%.0f')**

**output = imagine\_face\_marks(image,shape)**

**cv2.imshow("Image", output)**

**cv2.waitKey(0)**

**# load the input image2, resize it, and convert it to grayscale**

**image2 = cv2.imread("teja.jpg")**

**gray2 = cv2.cvtColor(image2, cv2.COLOR\_BGR2GRAY)**

**# detect faces in the grayscale image**

**rects2 = detector(gray2, 1)**

**# loop over the face detections**

**for (i, recb) in enumerate(rects2):**

**# determine the facial landmarks for the face region, then**

**# convert the landmark (x, y)-coordinates to a NumPy array**

**structure2 = predictor(gray2, recb)**

**structure2 = shape\_to\_numpy\_array(structure2)**

**shapeExb=np.concatenate((structure2,Extended\_points))**

**#print(structure2)**

**print(shapeExb)**

**np.savetxt("image2.txt",shapeExb,fmt ='%.0f')**

**outpub = imagine\_face\_marks(image2,structure2)**

**#cv2.imshow("Image2", outpub)**

**cv2.imshow("Image2", outpub)**

**cv2.waitKey(0)**

**button1 = Button(root,text="Open Two images",command=fun\_open\_picture,width = 50, height =2,relief =FLAT)**

**button1.pack(side=TOP,padx=5,pady=5,expand=True)**

**button2 = Button(root,text="detecting feature points of image",command=featurepoints ,width = 50, height =2,relief =FLAT)**

**button2.pack(side=TOP,padx=5,pady=5,expand=True)**

**button3 = Button(root,text="Delaunay triangulation 1 picture",command= fun\_del\_triangulation,width = 50, height =2,relief =FLAT)**

**button3.pack(side="left",padx=5,pady=5,expand=True)**

**button4 = Button(root,text="Delaunay Triangulation 2 picture",command=fun\_del\_triangulation2,width = 50, height =2,relief =FLAT)**

**button4.pack(side="right",padx=5,pady=5,expand=True)**

**button5 = Button(root,text="Morph",command=img\_morphing,width = 50, height =2,relief =FLAT)**

**button5.pack(side=TOP,padx=5,pady=5,expand=True)**

**root.mainloop()**