Multimedia Processing

PROJECT REPORT

17UCS001 - AAYUSH MISHRA

17UCS002 - ABHAY ARAVINDA

PART 1: Understanding The Problem

Take two images of the same size. Morph one image into another. This has to be done using a GUI that we have to create. This GUI should be used to mark control points on both the images. Delaunay Triangulation has to be applied using the points selected. The triangles thus formed should be warped using Affine Transformation. Finally, a set of intermediate images should be obtained that will show one image being morphed into the other image. These images should also be converted into a video.

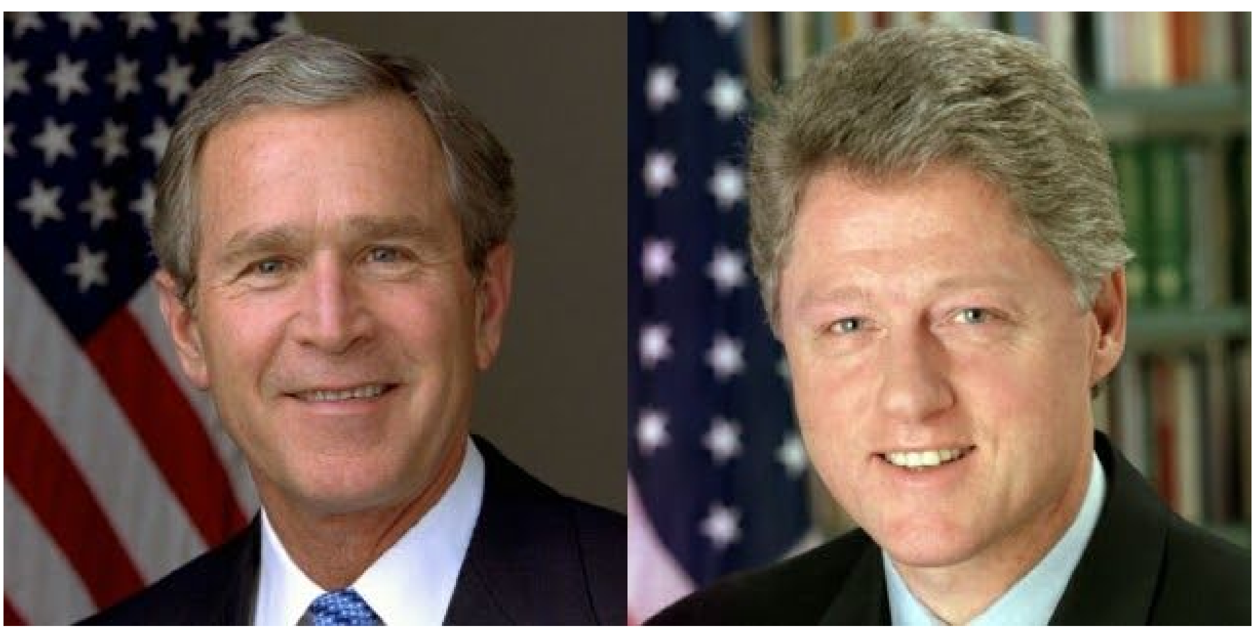


Fig 1. Sample images chosen for this project

PART 2: Steps Followed

We have segregated our entire program into 3 different Python Scripts. This was done in order to make the code easier for the user to understand.

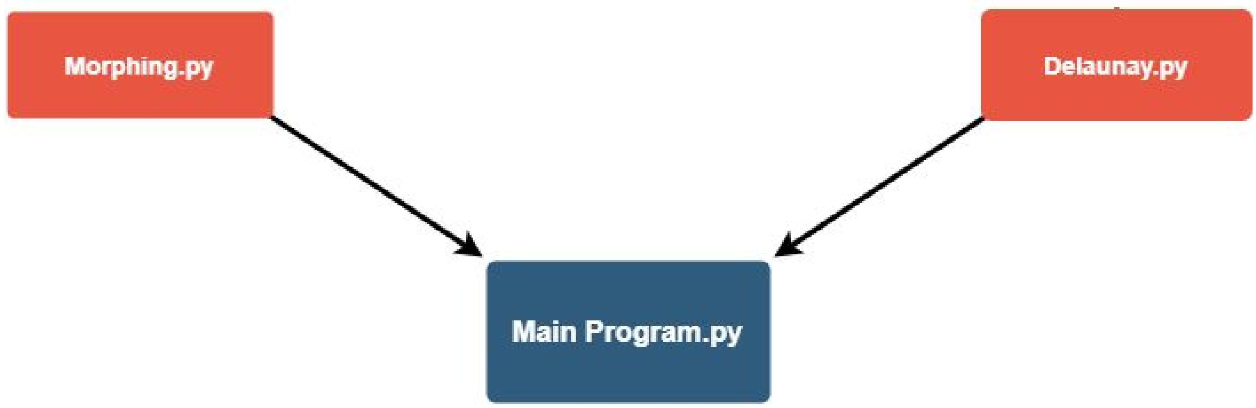


Fig 2. Relation of Program files

- ***Main_Program.py-***

  This is the main file and acts as the entry point of the program. The entire Graphical User Interface (GUI) and its functionalities are created over here. It imports the *Delaunay.py* file and *Morphing.py* file created by us. It takes the input of all the control points using mouse clicks. This file is also responsible for generating output frames and videos.

- ***Delaunay.py-***

  From an array of control points marked on the images, the program runs the Bowyer Watson Algorithm and returns a list of Delaunay triangles. It contains code written manually without the use of library functions.

- ***Morphing.py:-***

  Given a set of two triangles that act as control points, this file produces the intermediate triangles for the nth in-between frame.

PART 3: Algorithms/Flow of Programs

This section attempts to show a detailed analysis of what algorithm has been used in each program, the program flow and other important elements essential in the understanding of the project.

- **Main Program.py-**

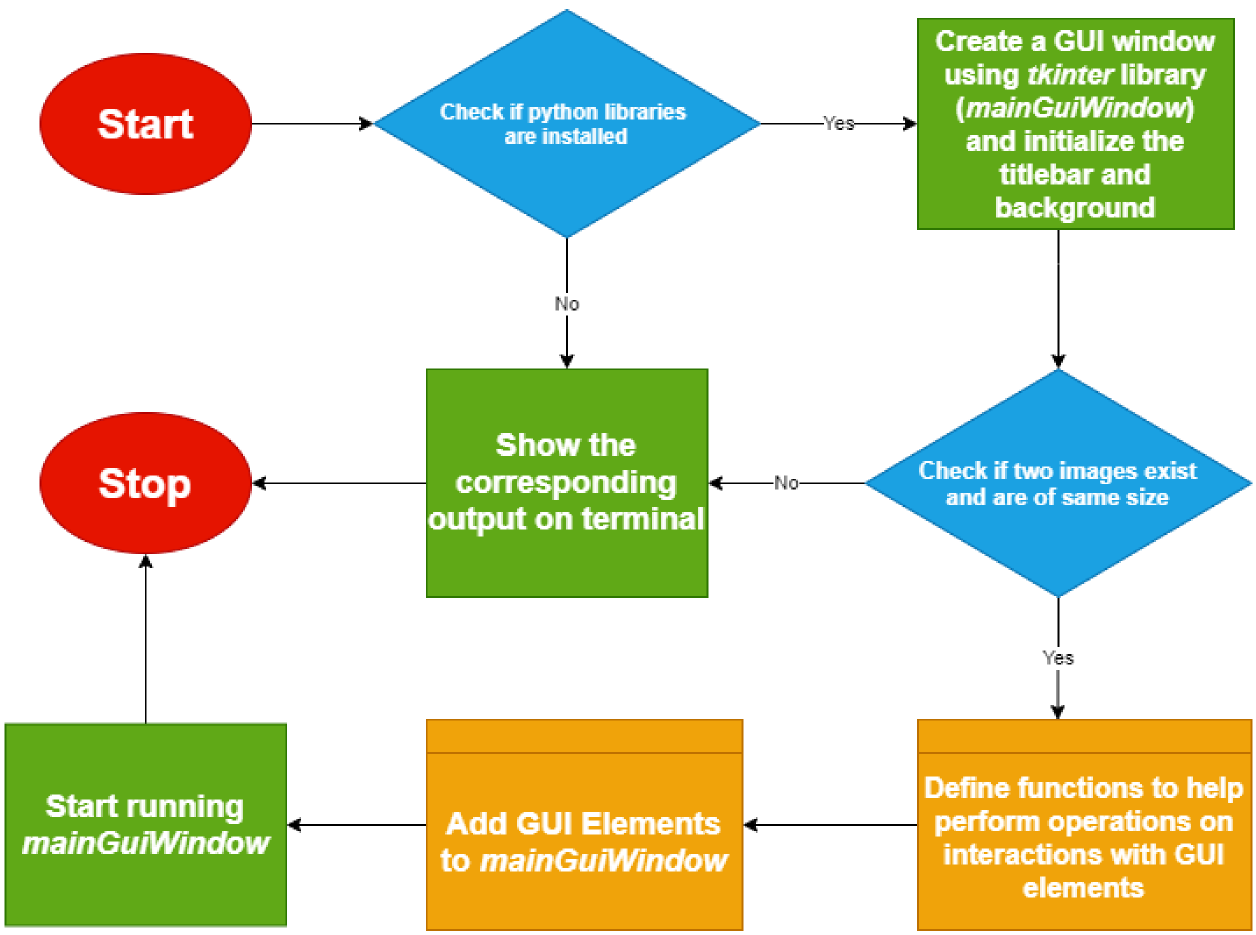


Fig 3. Explaining the flow of MainProgram.py

```python
def onToggleDelaunayTriangles(event):
    global showDelaunayTriangles
    global refreshCanvas1
    global refreshCanvas2
    showDelaunayTriangles.set(not(showDelaunayTriangles.get()))
    refreshCanvas1()
    refreshCanvas2()
```

Fig 4. onToggleDelaunayTriangles function

- Explanation of **onToggleDelaunayTriangles** function: The program contains a global variable ***showDelaunayTriangles***. The function sets **showDelaunayTriangles**:= !**showDelaunayTriangles.** It then refreshes the 2 canvases containing the images. (refer to the explanation of refreshCanvas functions)

```python
def changeNumberOfVideoFrames():
    global numberOfVideoFramesInput
    global totalFrames
    global submitNumberOfVideoFrames
    global statusBar
    global mainGuiWindow
    global morphInExecution
    if(morphInExecution): ...
    inputValue = numberOfVideoFramesInput.get()
    try:
        parsedValue = int(inputValue)
    except: ...
    if(parsedValue>0):
        totalFrames = parsedValue
        statusBar.config(text="Status Bar: Output will now have "+str(totalFrames)+" frames")
        mainGuiWindow.update()
    else: ...
```

Fig 5. changeNumberOfVideoFrames function

- Explanation of ***changeNumberOfVideoFrames*** function: The program checks if the value in the input field ***numberOfVideoFramesInput*** is an integer with value > 0. If yes, then it sets the global variable **totalFrames** to the value received. The function also updates the text of the status bar according to success or failure.

```python
def onFocusAwayFromNumberOfVideoFramesInput(event):
    global numberOfVideoFramesInput
    global totalFrames
    numberOfVideoFramesInput.delete(0,tk.END)
    numberOfVideoFramesInput.insert(0,totalFrames)
```

Fig 6. onFocusAwayFromNumberOfVideoFramesInput function

- Explanation for ***onFocusAwayFromNumberOfVideoFramesInput*** function: The value of the input field ***numberOfVideoFramesInput*** is reset to **totalFrames.** (Since user is no longer interested in it)

```
def changeDefaultColorForPoints():
    global defaultColorForPointsInput
    global defaultColoursForNewPoints
    global submitDefaultColorForPoints
    global statusBar
    global mainGuiWindow
    global refreshCanvas1
    global refreshCanvas2
    global statusBar
    global mainGuiWindow
    global morphInExecution
    if(morphInExecution): ...

    inputValue = defaultColorForPointsInput.get()
    validHexValues = '^[0-9a-fA-F]{6}$'
    if(re.match(validHexValues,inputValue)): ...
    else: ...

```

Fig 7. changeDefaultColorForPoints function

- Explanation of ***changeDefaultColorForPoints*** function: The program checks if the value in the input field **defaultColorForPointsInput** is a valid hex color code. If yes, then it sets the global variable ***defaultColoursForNewPoints*** to the value received. As a helpful indicator, it also sets the color of the entry ***defaultColorForPointsInput*** and the submit button ***submitDefaultColorForPoints*** to that particular value. The function also updates the text of the status bar according to success or failure. It then refreshes the canvases (The reasoning is, if a point in image1 is selected but its corresponding point in image 2 is not selected, the color value of point in image 1 should get updated)

```
def onFocusAwayFromDefaultColorForPointsInput(event):
    global defaultColorForPointsInput
    global defaultColoursForNewPoints
    defaultColorForPointsInput.delete(0,tk.END)
    defaultColorForPointsInput.insert(0,defaultColoursForNewPoints)
```

Fig 8. onFocusAwayFromDefaultColorForPointsInput function

- Explanation for ***onFocusAwayFromDefaultColorForPointsInput*** function: The value of the input field ***defaultColorForPointsInput*** is reset to **defaultColoursForNewPoints** (Since user is no longer interested in it)

```python
def refreshTable():
    global TableContainer
    global TableOfPoints

    TableContainer.destroy()
    TableContainer = tk.Frame(scrollable_frame,background="#FFFFAA")
    TableContainer.grid(row=5,column=0)
    if (len(TableOfPoints)==0):
        return
    tk.Label(TableContainer,text="Sl. No",padx=5,borderwidth=1, relief='solid').grid(row=0,column=0)
    tk.Label(TableContainer,text="Hex Color Code",padx=5,borderwidth=1, relief='solid').grid(row=0,column=1)
    tk.Label(TableContainer,text="Coordinates of Image 1",padx=5,borderwidth=1, relief='solid').grid(row=0,column=2)
    tk.Label(TableContainer,text="Coordinates of Image 2",padx=5,borderwidth=1, relief='solid').grid(row=0,column=3)
    tk.Label(TableContainer,text="",background="#FFFFAA").grid(row=0,column=4)

    numberOfEntries = 0
    for entry in TableOfPoints: ...
```

Fig 9. refreshTable function

- Explanation of **refreshTable** function: It uses the global variable ***TableOfPoints*** and creates a table in the **tkinter** frame ***TableContainer*.** The table thus formed shows the number of points, color of the points (in hex), coordinates of corresponding points in both images and has a Remove button that deletes an entry. The delete button triggers the ***deleteTableEntry*** function. We'd like to mention that we got stuck here and like to thank the StackOverflow user C.Nivs for fixing our bug at https://stackoverflow.com/questions/60710743/tkinter-passing-integer-by-value-intead-of-reference

```python
def deleteTableEntry(rowNumber):
    global TableOfPoints
    global morphInExecution
    global statusBar
    global mainGuiWindow
    if(morphInExecution):
        statusBar.config(text="Status Bar: Cannot remove entries. Morph is in execution")
        mainGuiWindow.update()
        return
    TableOfPoints.pop(rowNumber-1)
    refreshTable()
    refreshCanvas1()
    refreshCanvas2()
```

Fig 10. deleteTableEntry function

- Explanation of ***deleteTableEntry*** function: It takes the row number as a parameter and deletes that entry from the global variable ***TableOfPoints*.** It then invokes the ***refreshTable*** function. We felt it was not necessary to update the status bar once a row was deleted since the action was instantly visible. However, if the user tries to delete a row while the morphing is going on, then the status bar shows an error.

```python
def onClickOfImage1(event):
    global activeImage
    global coordinates
    global canvas1
    global radiusOfPoints
    global TableOfPoints
    global imageSize
    global morphInExecution
    global statusBar
    global mainGuiWindow
    if(morphInExecution): ...
    if(event.x<=0 and event.y<=0):
        return
    if(event.x>=imageSize[1]-1 and event.y<=0):
        return
    if(event.x<=0 and event.y>=imageSize[0]-1):
        return
    if(event.x>=imageSize[1]-1 and event.y>=imageSize[0]-1):
        return

    for entry in TableOfPoints: ...
    if(activeImage==1): ...
    else: ...
```

Fig 11. onClickOfImage1 function

- Explanation of **onClickOfImage1** function: It receives the coordinates control points based on mouse position relative to the image. (Since the image anchors to the northwest (top left corner), the mouse position is conveniently the x and y values that we need.) It throws an error in the status bar if we select a point that was already selected. If the global variable **activeImage** equals 1, then it updates the global variable **coordinates** (which is a temporary variable) to the values captured. It then sets **activeImage** to 2 and refreshes canvas1. If a point was already selected in image1 and image1 is again clicked before selecting the corresponding point in image2, then an error is shown in the status bar

```python
def onClickOfImage2(event):
    global activeImage
    global coordinates
    global TableOfPoints
    global imageSize
    global morphInExecution
    global statusBar
    global mainGuiWindow
    if(morphInExecution): ...
    if(event.x<=0 and event.y<=0):
        return
    if(event.x>=imageSize[1]-1 and event.y<=0):
        return
    if(event.x<=0 and event.y>=imageSize[0]-1):
        return
    if(event.x>=imageSize[1]-1 and event.y>=imageSize[0]-1):
        return

    for entry in TableOfPoints: ...
    if(activeImage==2): ...
    else: ...
```

Fig 12. onClickOfImage2 function

- Explanation of **onClickOfImage2** function: It works exactly like ***onClickOfImage1*** function. The only differences is, it refreshes both the canvases and sets **activeImage** to 2. It also creates a new entry in global variable ***TableOfPoints*** based on input and value stored in global variable **coordinates**

```python
def refreshCanvas1():
    global defaultColoursForNewPoints
    global url1
    global activeImage
    global image1
    global image1size
    global canvas1
    global radiusOfPoints
    global onClickOfImage1
    global TableOfPoints
    global showDelaunayTriangles
    canvas1.destroy()
    canvas1=tk.Canvas(ImageFramesContainer,height=image1size[0],width=image1size[1],borderwidth=0,highlightthickness=0)
    canvas1.create_image(1,1,image=image1,anchor='nw')
    canvas1.grid(row=0,column=0)
    canvas1.bind("<Button-1>",onClickOfImage1)
    for entry in TableOfPoints:
        canvas1.create_oval(entry[1][0]-radiusOfPoints,entry[1][1]-radiusOfPoints,entry[1][0]+radiusOfPoints,entry[1][1]+radiusOfPoints,fill="#"+entry[0])
    if(activeImage==2):
        canvas1.create_oval(coordinates[0][0]-radiusOfPoints,coordinates[0][1]-radiusOfPoints,coordinates[0][0]+radiusOfPoints,coordinates[0][1]+radiusOfPoints,fill="#"+defaultColoursForNewPoints)

    if(showDelaunayTriangles.get()):
```

Fig 13. refreshCanvas1 function

```python
def refreshCanvas2():
    refreshCanvas1()
    global defaultColoursForNewPoints
    global image2
    global image2size
    global canvas2
    global onClickOfImage2
    global showDelaunayTriangles
    global url2
    canvas2.destroy()
    canvas2=tk.Canvas(ImageFramesContainer,height=image1size[0],width=image1size[1],borderwidth=0,highlightthickness=0)
    canvas2.create_image(1,1,image=image2,anchor='nw')
    canvas2.grid(row=0,column=1)
    canvas2.bind("<Button-1>",onClickOfImage2)
    for entry in TableOfPoints:
        canvas2.create_oval(entry[2][0]-radiusOfPoints,entry[2][1]-radiusOfPoints,entry[2][0]+radiusOfPoints,entry[2][1]+radiusOfPoints,fill="#"+entry[0])

    if(showDelaunayTriangles.get()):
```

Fig 14. refreshCanvas2 function

- Explanation of ***refreshCanvas1*** and ***refreshCanvas2*** functions:

They destroy and recreate the **canvas1** and **canvas2** respectively. They then bind the canvases to functions ***onClickOfImage1*** and ***onClickOfImage2***. They then draw the images on the canvases. Then they iterate through ***TableOfPoints*** and plot points on the canvases. If **activeImage** equals 2, then there is an extra point in canvas 1 that is not in ***tableOfPoints*** (stored in temporary variable **coordinates**). This value is also plotted in **canvas1**. After plotting the points, the functions check the global variable **showDelaunayTriangles** and decide whether or not to draw Delaunay triangles. (Refer to explanation of Delaunay.py for how we generated the triangles)

```
def generateCorrespondingTriangles():
    global TableOfPoints
    global url1
    global url2
    setOfPointsInImage1=[]
    for entry in TableOfPoints:
        setOfPointsInImage1.append(entry[1])

    setOfPointsInImage2=[]
    for entry in TableOfPoints:
        setOfPointsInImage2.append(entry[2])

    imageSize = cv.imread(url1).shape
    triangleList1 = DT.findDelaunayTriangles(setOfPointsInImage1,imageSize[0],imageSize[1])
    triangleList2 = DT.findDelaunayTriangles(setOfPointsInImage2,imageSize[0],imageSize[1])

    if(len(triangleList1) != len(triangleList2)):
        return -1

    serializedTriangleList1=[]
    for triangle in triangleList1:

    serializedTriangleList2=[]
    for triangle in triangleList2:

    mapping=[]
    index1 = 0
    for entry1 in serializedTriangleList1:

    if(len(mapping) != len(triangleList2)):
        return -2

    correspondingTrianglesList=[]
    for entry in mapping:
    return correspondingTrianglesList
```

Fig 15.generateCorrespondingTriangles function

- Explanation of **generateCorrespondingTriangles** function: It takes the global variable **tableOfPoints** and extracts points in image1 and image2. It then calculates Delaunay triangles in both images. The coordinates of triangles are replaced with indexes of those coordinates in **tableOfPoints**. The Delaunay triangles are then sorted based on the indexes in both images. Using sorted representation of triangles, a mapping is created wherein 2 triangles are mapped if and only if all 3 points of both triangles correspond (in any order). The mapping is then inverted to get a set of corresponding Delaunay triangles (while preserving order of corresponding vertices). The corresponding list of triangles obtained is returned by the function. It instead returns -1 and -2 respectively if the number of Delaunay triangles formed do not match or the thus formed Delaunay triangles do not correspond.

```python
def executeMorph():
    global generateCorrespondingTriangles
    global url1
    global url2
    global statusBar
    global morphInExecution
    global activeImage
    global refreshCanvas1
    global refreshCanvas2
    global statusBar
    global mainGuiWindow
    global totalFrames
    activeImage=1
    refreshCanvas1()
    refreshCanvas2()
    if(morphInExecution): ...
    else:
        morphInExecution=True
    listOfTriangles = generateCorrespondingTriangles()
    if(listOfTriangles == -1): ...
    if(listOfTriangles == -2): ...

    if (len(listOfTriangles)==0): ...

    outputVideo1 = cv.VideoWriter('./OutputFolder/WithBackground.avi', cv.VideoWriter_fourcc(*'MP42'), float(1), (imageSize[1], imageSize[0]))
    outputVideo2 = cv.VideoWriter('./OutputFolder/WithoutBackground.avi', cv.VideoWriter_fourcc(*'MP42'), float(1), (imageSize[1], imageSize[0]))

    for i in range (0,totalFrames+1): ...
    statusBar.config(text="Status Bar: Adding final formatting to video")
    mainGuiWindow.update()
    outputVideo1.release()
    outputVideo2.release()


    statusBar.config(text="Status Bar:")
    mainGuiWindow.update()
    morphInExecution=False
```

Fig 16. executeMorph function

- Explanation of ***executeMorph*** function: It first gets the list of corresponding triangles using ***generateCorrespondingTriangles*** function. If the function returns a negative value (error value) or if insufficient points are selected then an error is shown in the status bar. Then for each frame, first a background is generated(one by converting all pixels to black and other by interpolating the values). Then triangles are morphed (refer to Morph.py explanation) for each frame and placed on both the backgrounds. Then the frames are appended to videos **outputVideo1** (Interpolated Background) and **outputVideo2** (Black Background). All the intermediate images and the final videos are stored in the output folder. Here's how our GUI works, the program runs an infinite loop continuously refreshing the screen. But since the morphing process takes some time, we added a few lines in between to refresh the screen (otherwise the GUI would appear to lag). We also decided to show the progress on the status bar to keep the user informed. The last change we made was, we added a global variable **morphInExecution.** The variable acts as a lock so that the user does not alter the points or number of frames or start another morph while morph is in execution. We made most of the function first check if **morphInExecution** and if true, throw an error in the status bar instead of executing.

• **Delaunay.py-**

This part of the program handles everything concerned with Delaunay Triangles.

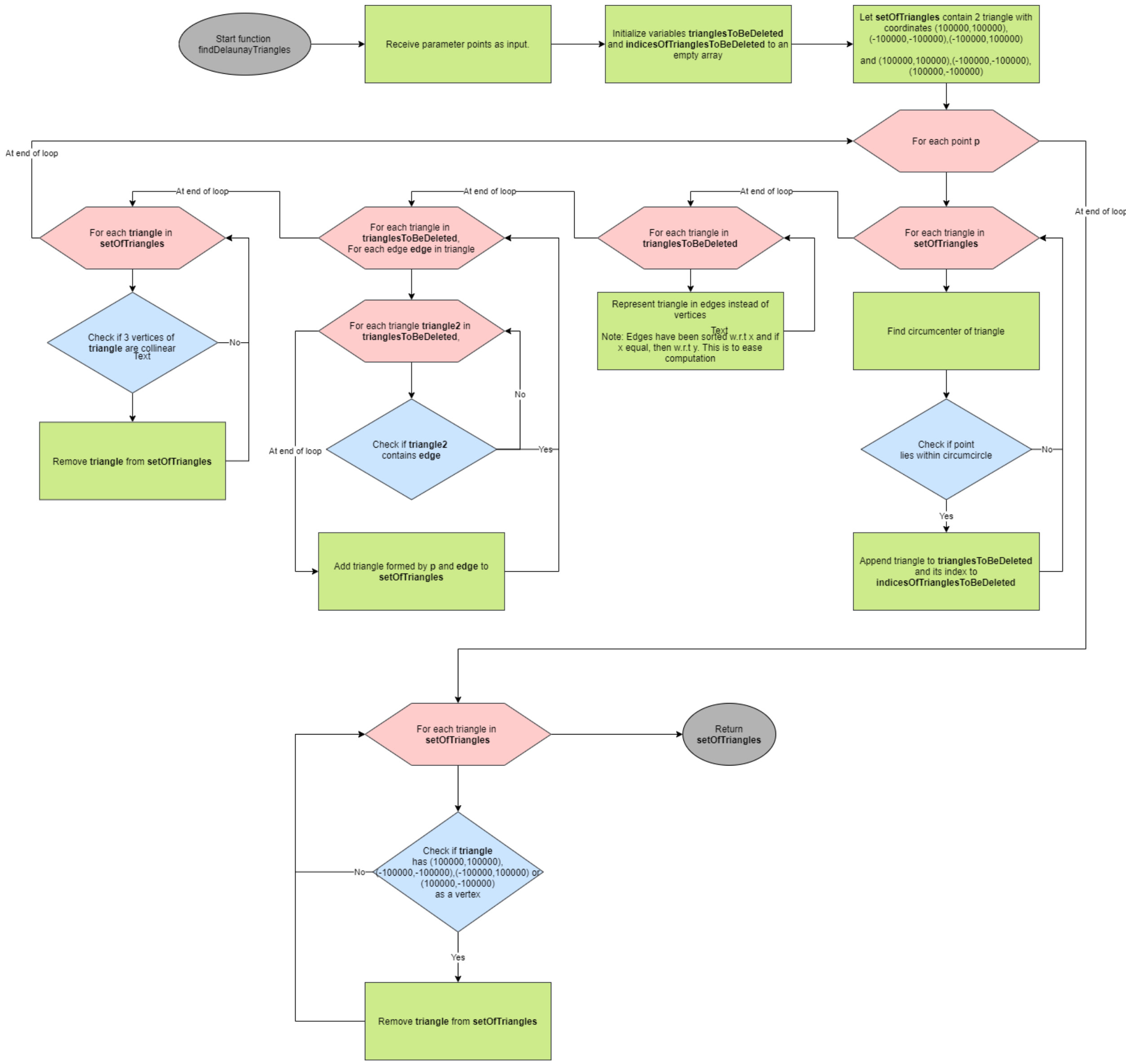
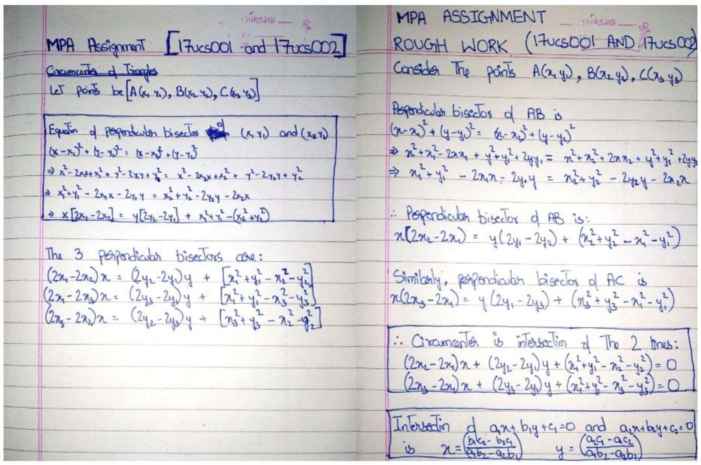


Fig 17. Flowchart representing the working of Delaunay.py

We made a few changes to the **Bowyer-Algorithm** given on Wikipedia:

- Instead of storing edges in an array called **polygon,** we directly added them to ***setOfTriangles.***
- The given algorithm sometimes resulted in 3 collinear points forming being included in ***setOfTriangles***. We identified and got rid of these triangles.

- By our initial understanding, we by default included the 4 corners of the image in the set of control points. The reasoning was, if those 4 points are included, then every pixel will lie in some Delaunay triangle. However, we realized that if the triangle was rotated by 180 degrees, then the Delaunay triangles thus formed would not correspond. So we got rid of the default corner points. (However, the user can still input them)


MPA Assignment [17ucs001 and 17ucs002]

Circumcenter of Triangles

Let points be [$A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$]

Equation of perpendicular bisector of (x_1, y_1) and (x_2, y_2)

$(x-x_1)^2 + (y-y_1)^2 = (x-x_2)^2 + (y-y_2)^2$

$\Rightarrow x^2 - 2xx_1 + x_1^2 + y^2 - 2y_1y + y_1^2 = x^2 - 2x_2x + x_2^2 + y^2 - 2y_2y + y_2^2$

$\Rightarrow x_1^2 + y_1^2 - 2x_1x - 2y_1y = x_2^2 + y_2^2 - 2y_2y - 2x_2x$

$\Rightarrow x[2x_1 - 2x_2] = y[2y_2 - 2y_1] + x_1^2 + y_1^2 - (x_2^2 + y_2^2)$

The 3 perpendicular bisectors are:

$(2x_1 - 2x_2)x = (2y_2 - 2y_1)y + [x_1^2 + y_1^2 - x_2^2 - y_2^2]$

$(2x_1 - 2x_3)x = (2y_3 - 2y_1)y + [x_1^2 + y_1^2 - x_3^2 - y_3^2]$

$(2x_3 - 2x_2)x = (2y_2 - 2y_3)y + [x_3^2 + y_3^2 - x_2^2 - y_2^2]$

MPA ASSIGNMENT

ROUGH WORK (17ucs001 AND 17ucs002)

Consider the points $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$

Perpendicular bisector of AB is

$(x-x_1)^2 + (y-y_1)^2 = (x-x_2)^2 + (y-y_2)^2$

$\Rightarrow x^2 + x_1^2 - 2xx_1 + y^2 + y_1^2 + 2yy_1 = x^2 + x_2^2 + 2xx_2 + y^2 + y_1^2 + 2yy_2$

$\Rightarrow x_1^2 + y_1^2 - 2x_1x - 2y_1y = x_2^2 + y_2^2 - 2y_2y - 2x_2x$

$\therefore$ Perpendicular bisector of AB is:

$x(2x_2 - 2x_1) = y(2y_1 - 2y_2) + (x_2^2 + y_2^2 - x_1^2 - y_1^2)$

Similarly, perpendicular bisector of AC is

$x(2x_3 - 2x_1) = y(2y_1 - 2y_3) + (x_3^2 + y_3^2 - x_1^2 - y_1^2)$

$\therefore$ Circumcenter is intersection of the 2 lines:

$(2x_2 - 2x_1)x + (2y_2 - 2y_1)y + (x_1^2 + y_1^2 - x_2^2 - y_2^2) = 0$

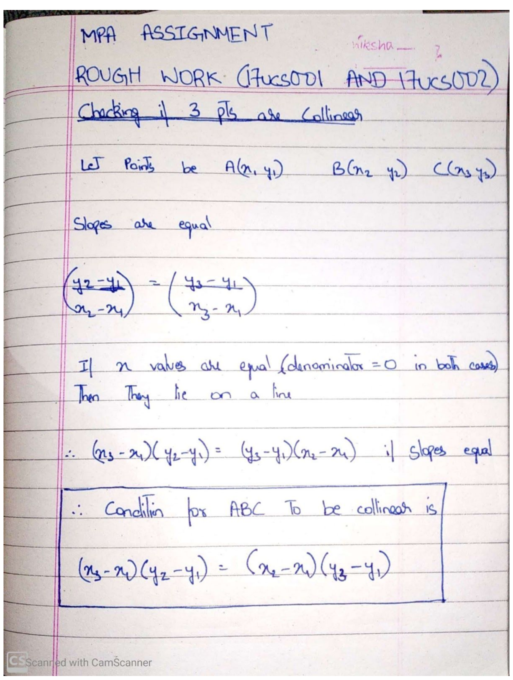
$(2x_3 - 2x_1)x + (2y_3 - 2y_1)y + (x_1^2 + y_1^2 - x_3^2 - y_3^2) = 0$

Intersection of $a_1x + b_1y + c_1 = 0$ and $a_2x + b_2y + c_2 = 0$ is $x = \left(\frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1}\right)$ $y = \left(\frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}\right)$

Fig 18. Formulas used for finding circumcenters


```python
def findCircumcenter(triangle):
    x1 = triangle[0]
    y1 = triangle[1]
    x2 = triangle[2]
    y2 = triangle[3]
    x3 = triangle[4]
    y3 = triangle[5]
    a1 = float(2*x2 - 2*x1)
    b1 = float(2*y2 - 2*y1)
    c1 = float(x1*x1 + y1*y1 - x2*x2 - y2*y2)
    a2 = float(2*x3 - 2*x1)
    b2 = float(2*y3 - 2*y1)
    c2 = float(x1*x1 + y1*y1 - x3*x3 - y3*y3)
    x = (b1*c2 - b2*c1)/(a1*b2 - a2*b1)
    y = (a2*c1 - a1*c2)/(a1*b2 - a2*b1)
    return [x,y]
```

Fig 19. Code based on Formulas


MPA ASSIGNMENT

ROUGH WORK (17ucs001 AND 17ucs002)

Checking if 3 pts are Collinear

Let Points be $A(x_1, y_1)$ $B(x_2, y_2)$ $C(x_3, y_3)$

Slopes are equal

$\left(\frac{y_2 - y_1}{x_2 - x_1}\right) = \left(\frac{y_3 - y_1}{x_3 - x_1}\right)$

If x values are equal (denominator = 0 in both cases)

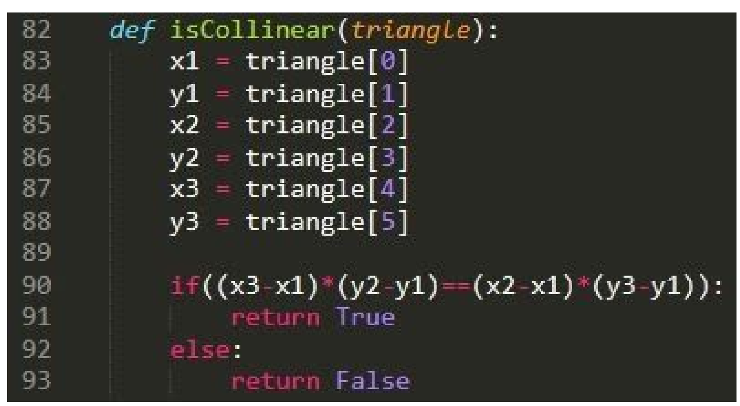
Then they lie on a line

$\therefore (x_3 - x_1)(y_2 - y_1) = (y_3 - y_1)(x_2 - x_1)$ if slopes equal

$\therefore$ Condition for ABC to be collinear is

$(x_3 - x_1)(y_2 - y_1) = (x_2 - x_1)(y_3 - y_1)$

Fig 20. Formulas used for checking collinear points


```python
def isCollinear(triangle):
    x1 = triangle[0]
    y1 = triangle[1]
    x2 = triangle[2]
    y2 = triangle[3]
    x3 = triangle[4]
    y3 = triangle[5]

    if((x3-x1)*(y2-y1)==(x2-x1)*(y3-y1)):
        return True
    else:
        return False
```

Fig 21. Code based on Formulas

- **Morphing.py-**

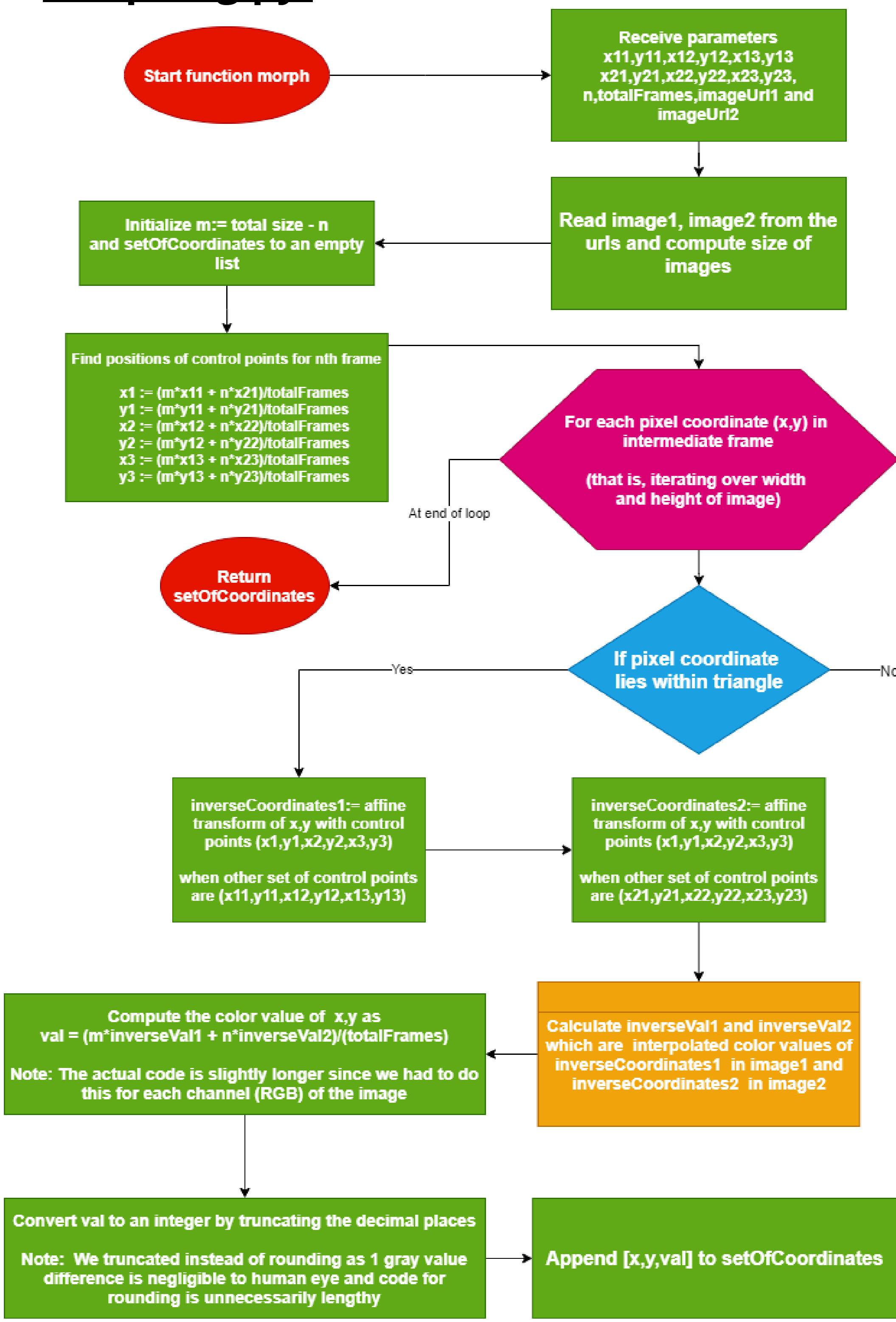


Fig 22. Flowchart representing the working of Morphing.py

This part of the Project deals with the Morphing of the images and everything associated with it.

```python
def inverseColorValueInterpolated(coordinates,image):
    #Returns interpolated value for fractional coordinates.
    x = coordinates[0]
    y = coordinates[1]
    x1 = int(x)
    y1 = int(y)
    x2 = x1+1
    y2 = y1+1

    value=[]
    numberOfChannels = len(image[y1][x1])
    for i in range(0,numberOfChannels):

    return value

```

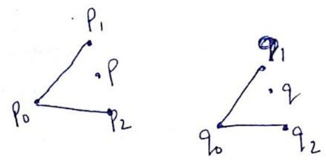
Fig 23. inverseColorValueInterpolated function

To calculate the interpolated color value for coordinates **(x,y)** where **x** and **y** are float values, we divided it into 4 cases:

- **Case 1:** If **x** and **y** are both integers. Then return the value of **(x,y).**
- **Case 2:** If only **y** is integer, then find **x1:=truncation(x)** and **x2:=x1+1.** Find return interpolation of **(x1,y)** and **(x2,y).**
- **Case 3:** If only, **x** is an integer, then similar to **case 2**, return the interpolation of **(x,y1)** and **(x,y2).**
- **Case 4:** If neither **x** nor **y** is an integer, find ***interpolatedXValueFory1*** by interpolating **(x1,y1)** and **(x2,y1).**
  Then find ***interpolatedXValueFory2*** by interpolating **(x1,y2)** and **(x2,y2).**
  Then return the interpolation of these 2 values.

MPA Assignment Calculations

(17ucs001 & 17ucs002)



$$(p - p_0) = \alpha(p_1 - p_0) + \beta(p_2 - p_0)$$
$$(q - q_0) = \alpha(q_1 - q_0) + \beta(q_2 - q_0)$$

Let
$$q - q_0 = A$$
$$q_1 - q_0 = B$$
$$q_2 - q_0 = C$$

$$\boxed{p = p_0 + \alpha(p_1 - p_0) + \beta(p_2 - p_0)}$$

$$A_x = \alpha B_x + \beta C_x$$
$$A_y = \alpha B_y + \beta C_y$$

$$A_x B_y = \alpha B_x B_y + \beta C_x B_y$$
$$A_y B_x = \alpha B_y B_x + \beta C_y B_x \Rightarrow \boxed{\beta = \frac{A_x B_y - A_y B_x}{C_x B_y - C_y B_x}}$$

$$A_x C_y = \alpha B_x C_y + \beta C_x C_y$$
$$A_y C_x = \alpha B_y C_x + \beta C_y C_x \Rightarrow \boxed{\alpha = \frac{A_x C_y - A_y C_x}{B_x C_y - B_y C_x}}$$

Fig 24. Formulas used for Affine Transform

```python
def affineTransform(p0x,p0y,p1x,p1y,p2x,p2y,q0x,q0y,q1x,q1y,q2x,q2y,qx,qy):
    #Affine Transform for 2 triangles and 2 points (refer calculations)
    beta = ((qx-q0x)*(q1y-q0y)-(qy-q0y)*(q1x-q0x))/((q2x-q0x)*(q1y-q0y)-(q2y-q0y)*(q1x-q0x))
    alpha = ((qx-q0x)*(q2y-q0y)-(qy-q0y)*(q2x-q0x))/((q1x-q0x)*(q2y-q0y)-(q1y-q0y)*(q2x-q0x))
    px = p0x + alpha*(p1x-p0x) + beta*(p2x-p0x)
    py = p0y + alpha*(p1y-p0y) + beta*(p2y-p0y)
    return [px,py]

```

Fig 25. Code for Affine Transform

PART 4: Visual Walkthrough

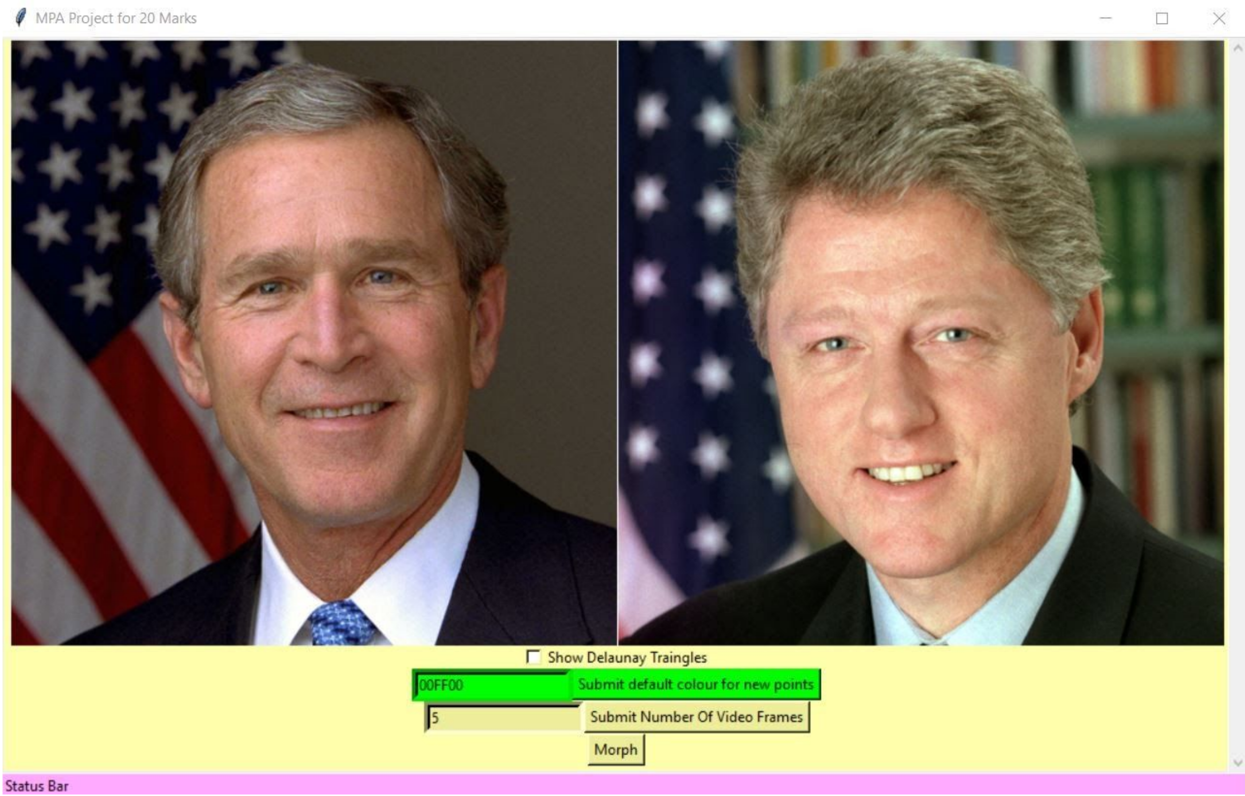


Fig 26. The initial view of the GUI

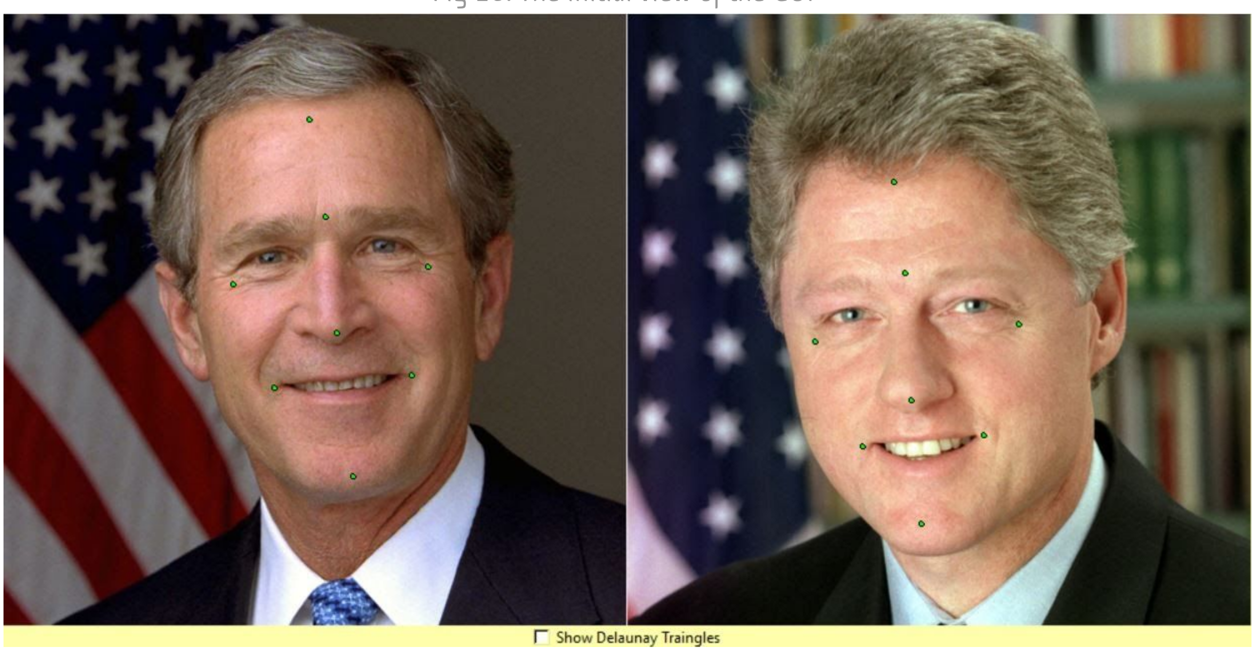


Fig 27. Feature Points Marked

Sl. No	Hex Color Code	Coordinates of Image 1	Coordinates of Image 2	
1	00FF00	(185 , 226)	(151 , 272)	REMOVE
2	00FF00	(341 , 212)	(314 , 258)	REMOVE
3	00FF00	(218 , 309)	(189 , 356)	REMOVE
4	00FF00	(281 , 380)	(236 , 418)	REMOVE
5	00FF00	(259 , 172)	(223 , 217)	REMOVE
6	00FF00	(328 , 299)	(286 , 347)	REMOVE
7	00FF00	(268 , 265)	(228 , 319)	REMOVE
8	00FF00	(246 , 94)	(214 , 144)	REMOVE

Fig 28. Table for Feature Points present on GUI

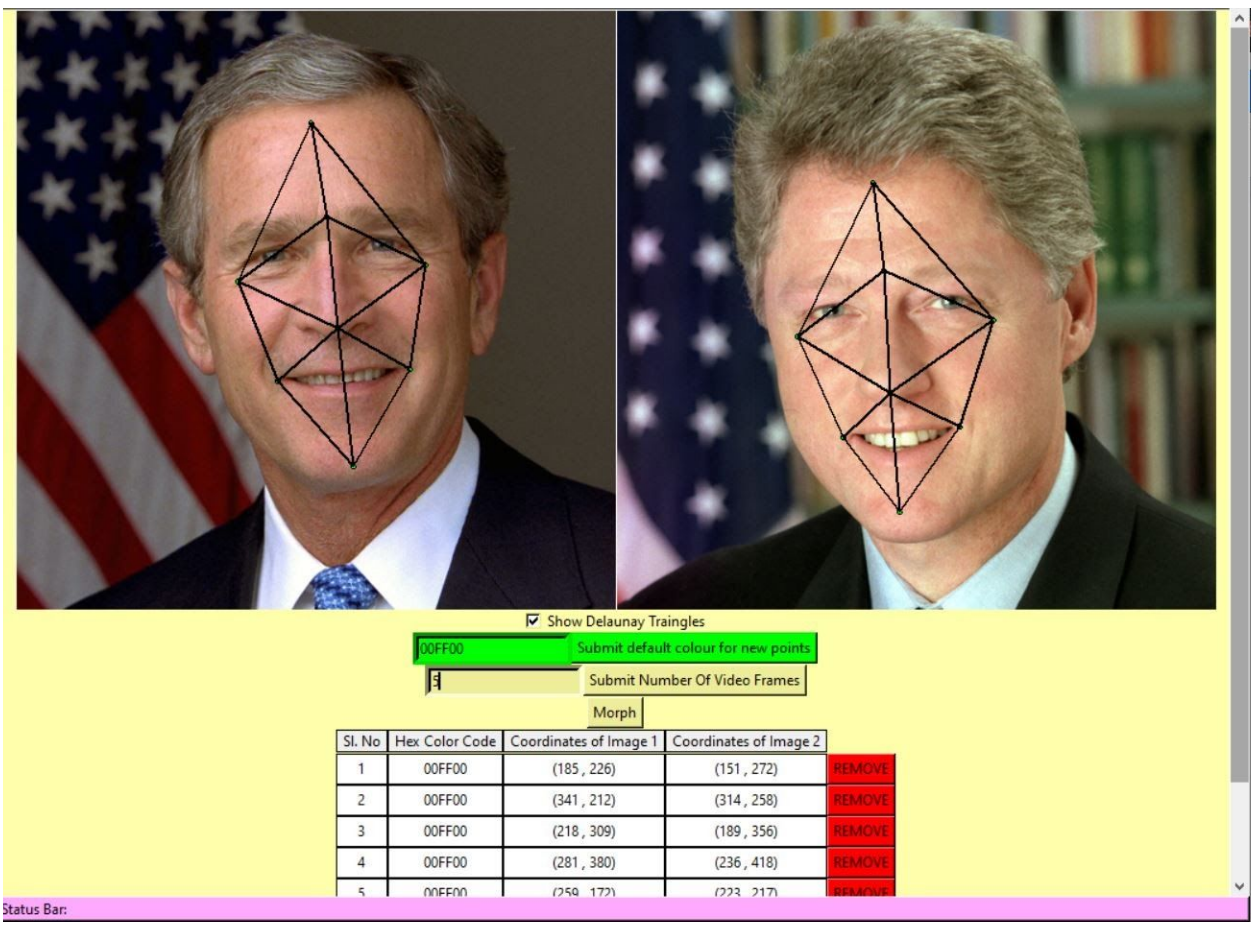


Fig 29. Corresponding Delaunay Triangles mapped for given feature points

Fig 30. After clicking the morph button, the status of morphing can be seen here

Fig 31. Intermediate images with background

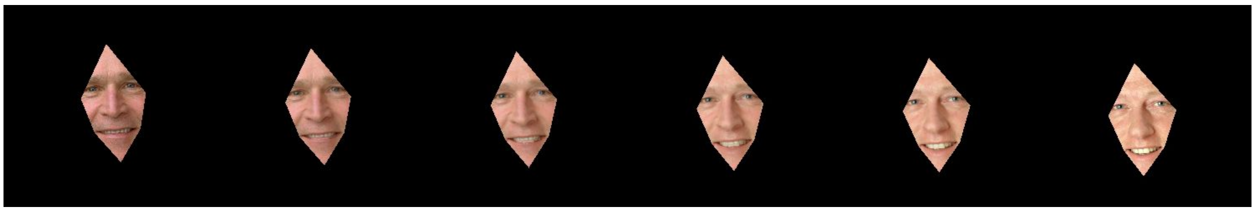


Fig 32. Intermediate images without background

PART 5: List of references

1. *Bowyer Watson Algorithm Pseudocode:*
   https://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm
2. *For locating whether a point lies inside a triangle or not:*
   https://www.youtube.com/watch?v=H9qu9Xptf-w
3. *For fixing bug in our program:*
   https://stackoverflow.com/questions/60710743/tkinter-passing-integer-by-value-inte
   ad-of-reference
4. *For creating GUI:*
   https://www.youtube.com/watch?v=YXPyB4XeYLA
5. *For scrollbar:*
   https://www.youtube.com/watch?v=XkCbinbgbdw
6. *For removing border of images(in GUI):*
   https://stackoverflow.com/questions/43880224/python-tkinter-how-to-remove-the-
   border-around-a-frame
7. *For creating scrollbar:*
   https://blog.tecladocode.com/tkinter-scrollable-frames/
   https://stackoverflow.com/questions/16820520/tkinter-canvas-create-window
8. *For generating video from opencv:*
   https://medium.com/@enriqueav/how-to-create-video-animations-using-python-an
   d-opencv-881b18e41397