**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY**

**UNIVERSITY OF INFORMATION TECHNOLOGY**

**Logo

Description automatically generatedFACULTY OF COMPUTER SCIENCE**

**FINAL REPORT**

**INTRODUCTION TO COMPUTER VISION**

*Project:* **Virtual Singer**

**……..**

|  |  |
| --- | --- |
| ***Instructor:*** | ***PhD:*** Nguyễn Vĩnh Tiệp |
| ***Class:*** | CS231.N11.KHCL |
| ***Group members:*** | |
| Bùi Tuấn Kiệt | 20521493 |
| Trần Đình Khôi | 20521482 |
| Vũ Quốc Thái Bình | 20521119 |
| Bùi Viết Đạt | 20521162 |

**Ho Chi Minh City, 01/2023**

**TABLE OF CONTENT**

[CHAPTER 1: INTRODUCTION 1](#_Toc124883999)

[1.1 Topic Introduction 1](#_Toc124884000)

[1.2 Topic Goal 1](#_Toc124884001)

[1.3 Related Work 1](#_Toc124884002)

[1.3.1 Landmark Point 1](#_Toc124884003)

[1.3.2 Delaunay Triangulation 3](#_Toc124884004)

[1.3.3 Affine Transformation 4](#_Toc124884005)

[1.3.4 Seamless Clone 6](#_Toc124884006)

[CHAPTER 2: METHODS 8](#_Toc124884007)

[2.1 Face Swap Replace Facial Features 8](#_Toc124884008)

[2.2 Face Swap Delaunay 12](#_Toc124884009)

[2.3 Face Swap Using OPENGL 16](#_Toc124884010)

[CHAPTER 3: RESULT & COMPARISON 18](#_Toc124884011)

[3.1 Result 18](#_Toc124884012)

[3.2 Comparison 19](#_Toc124884013)

# CHAPTER 1: INTRODUCTION

## Topic Introduction

Face swapping is a task to switch the person-identity of a given face image with another, preserving other attributes like facial expressions, head poses, and backgrounds. The task has been highlighted for its wide use of real-world applications, such as anonymization in privacy protection and the creation of new characters in the entertainment industry.

## Topic Goal

The aim of this project is to implement an end-to-end pipeline to swap faces in a video just like Snapchat's face swap filter or Tiktok filter. There are two methods and four distinct implements to the project. Each is explained in detail below.

## Related Work

### Landmark Point

In morphometrics, landmark point or shortly landmark is a point in a shape object in which correspondences between and within the populations of the object are preserved. In other disciplines, landmarks may be known as vertices, anchor points, control points, sites, profile points, 'sampling' points, nodes, markers, fiducial markers, etc. Landmarks can be defined either manually by experts or automatically by a computer program.

In this project, we mainly use dlib and mediapipe library to find out the landmark point in image.

* Dlib:

It‘s a landmark’s facial detector with pre-trained models, the dlib is used to estimate the location of 68 or 81 coordinates (x, y) that map the facial points on a person’s face like image below. Those 2 models are :

shape\_predictor\_68\_face\_landmarks.dat shape\_predictor\_81\_face\_landmarks.dat

|  |  |
| --- | --- |
|  | A picture containing person, person, spectacles  Description automatically generated |
| *68\_face\_landmarks* | *81\_face\_landmarks* |

* Mediapipe:

MediaPipe Face Mesh is a solution that estimates 468 3D face landmarks in real-time. It is assumed that the input camera frames are observed by exactly this virtual camera located at the space origin and pointed in the negative direction of the Z-axis.

|  |  |
| --- | --- |
| Face Mesh | mediapipe | deep learning - YouTube | Each landmark is composed of x, y and z. x and y are normalized to [0.0, 1.0] by the image width and height respectively. |
| *MediaPipe Face Mesh* | |

### Delaunay Triangulation

In mathematics and computational geometry, a Delaunay triangulation (also known as a Delone triangulation) for a given set P of discrete points in a general position is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P). Delaunay triangulations maximize the minimum of all the angles of the triangles in the triangulation.

|  |
| --- |
| Background pattern  Description automatically generated |
| *A Delaunay triangulation in the plane with circumcircles shown* |

* Advantages of Delaunay triangulation:

The figure below shows an example of what happens when a set of points is connected in an arbitrary manner versus one that conforms to the Delaunay criterion.

|  |  |
| --- | --- |
|  |  |
| *Normal Triangulation* | *Delaunay Triangulation* |

One important characteristic of the Delaunay is that it tends to make "good" choices about which vertices are connected to each other. In both the examples above, vertices were produced using a random process and added to the triangulations one at a time. In the triangulation on the left, the vertices were linked together based solely on the order in which they were received. The assembly process made no effort to optimize the arrangement. That's why some of the vertices that are located close together ended up being separated by an intervening edge. In contrast, the triangulation on the right established links based on the Delaunay criterion. As you can see, it did a much better job of connecting vertices to their neighbors so that proximity relationships were preserved.

Another consequence of the Delaunay is that the triangles it produces are, on the whole, more nearly equilateral than those in non-optimal triangulations. For interpolation and data modeling purposes, these "robust" triangles have an advantage over "skinny" triangles in that they tend to provide a more uniform representation of the surface. In particular, the robust triangles result in more gradual changes in slope across the edges than the skinny triangles. And because there is a smoother transition across edges, interpolated surfaces from Delaunay conforming triangulations tend to have better continuity properties than those from non-optimal triangulations.

### Affine Transformation

An affine transformation is any transformation that preserves collinearity *(i.e., all points lying on a line initially still lie on a line after transformation)* and ratios of distances *(e.g., the midpoint of a line segment remains the midpoint after transformation)*. In this sense, affine indicates a special class of projective transformations that do not move any objects from the affine space R3 to the plane at infinity or conversely. An affine transformation is also called an affinity.

Geometric contraction, expansion, dilation, reflection, rotation, shear, similarity transformations, spiral similarities, and translation are all affine transformations, as are their combinations. In general, an affine transformation is a composition of rotations, translations, dilations, and shears.

In general, the affine transformation can be expressed in the form of a linear transformation followed by a vector addition as shown below:

|  |  |
| --- | --- |
| Text  Description automatically generated | Since the transformation matrix (M) is defined by 6 (2x3 matrix) constants, thus to find this matrix we first select 3 points in the input image and map these 3 points to the desired locations in the unknown output image according to the use-case as shown below (This way we will have 6 equations and 6 unknowns and that can be easily solved). |
| *Affine Transformation* | |

Diagram

Description automatically generated

Below is the output image. Here, left image represents the original image while the right one is the transformed mirror iamge.

Logo, icon

Description automatically generated with medium confidence

### Seamless Clone

‘Seamless Clone’ is the exciting new feature that has been recently introduced in OpenCV. With this function, we can do the color correction effectively in a much easier way. Here is the function:

**cv2.seamlessClone**(Mat src, Mat dst, Mat mask, Point center, Mat output, int flags)

***Below are the two figures***

|  |  |
| --- | --- |
| An "I Love You" Ticket. | wood-texture |
| *Figure 1: An “I love you” ticket* | *Figure 2: Wood Texture* |

***Result***

|  |  |
| --- | --- |
| **Normal Cloning (NORMAL\_CLONE)** | **Mixed Cloning**  **(MIXED\_CLONE)** |
|  |  |
| *Figure 3: OpenCV Normal Clone Example* | *Figure 4: OpenCV Mixed Cloning Example* |

If we use Normal Cloning by using the NORMAL\_CLONE flag, we will get the result shown in Figure 3. Now we did not use a good mask and you can see excessive smoothing between the words “I” and “Love”, and between “you” and “Paa”. We could have created a rough mask and improved the result by using Mixed Cloning.

In Normal Cloning the texture of the source image is preserved in the cloned region. In Mixed Cloning, the texture of the cloned region is determined by a combination of the source and the destination images. Mixed Cloning does not produce smooth regions because it picks the dominant texture between the source and destination images. The result of mixed cloning is shown in Figure 4. Notice the texture is no longer smooth between “I” and “Love”, and “you” and “Paa”.

However, in face swapping, we do not use the Mixed Cloning as it might destroy the content of the face. We just want to adjust the color of the source skin to fitting the destination face. So, in our method, we use the Normal Cloning in the entire method.

# CHAPTER 2: METHODS

## 2.1 Face Swap Replace Facial Features

Face swap method to automatically replace facial features of a face, with the facial features from a second image of a face. This process breaks down into four steps:

* Detecting facial landmarks
* Fit the face in the first image to second image
* Making a mask to show facial features of the first image on the second image
* Color correction skin tone of two pictures

***Methodology***

First we have two pictures:

* The picture which has the face we want to blend into the other picture called source image (source for short)
* The picture which has the face we want the face swapped called destination image (destination for short)

|  |  |
| --- | --- |
| A person with a beard  Description automatically generated with low confidence | A picture containing person, person, suit, wearing  Description automatically generated |
| *Source image* | *Destination image* |

***Step 1:*** Finding facial landmarks

Using Dlib library and pre-trained model from [dlib sourceforge repository](https://sourceforge.net/projects/dclib/files/dlib/v18.10/shape_predictor_68_face_landmarks.dat.bz2/download)

The function will return 682 numpy array which contains the x, y coordinates of particular facial feature of the input image

|  |  |
| --- | --- |
|  |  |
| *Landmark point in source image* | *Landmark point in destination image* |

***Step 2:*** Rotate, translate and scale

Next we will need to rotate, translate, scale and stack two images on each other where the face of the source image fit a close as possible to the destination image. This will be break down into 3 parts:

1. The center-of-mass is subtracted to locate each group of points at the origin



1. Next the points are scaled according to the standard deviation of the points so that the faces are the same size



1. Finally using singular value decomposition to get the rotation that best matches the points on top each other

A picture containing text

Description automatically generated

Then this will returned as affine transformation matrix that can be plugged into **cv2.warpAffine** function

***Step 3:*** Making a mask

To show facial feature of source image on the destination image we will need a mask

|  |
| --- |
|  |
| *Collect points that involve eyes, mouth, nose and eyebrow* |

Next we will draw a mask by using **cv2.convexHull** where the white area will be from the source and black will be from the destination. The mask will be blur by **cv2.GaussianBlur**

|  |  |
| --- | --- |
|  |  |
| *cv2.convexHull* | *cv2.GaussianBlur* |

***Step 4:*** Color correction

|  |  |
| --- | --- |
| Image won’t really look nature when overlay facial features right now  The issue is that differences in skin tone and lighting in two images |  |

We will try to fix that by a using a method call RGB scaling color correction



We will divide the source by a gaussian blur of source and then multiplying by a gaussian blur of destination

The idea here is that of a RGB scaling colour-correction, but instead of a constant scale factor across all of the image, each pixel has its own localised scale factor. We can see that that the face color from source will be scale to the color from destination

|  |  |
| --- | --- |
|  |  |
| After applied all the mask this is the final result | |
|  | |

## 2.2 Face Swap Delaunay

***Step 1:*** Find landmark points of both images

Use the pre-trained models of dlib or mediapipe library to detect the facial landmark points on source and destination image. In the example, I use model shape\_predictor\_68\_face\_landmarks.dat to illustrate those point

|  |  |
| --- | --- |
|  |  |
| *Landmark point in source image* | *Landmark point in destination image* |

***Step 2:*** Triangulation source and destination image

The reason why we need to divide the face into triangles is that we can not just cut out the face from the source image and put it into the destination image as they have different size and perspective.

Also we can not change its size and perspective right away because the face would lose the original proportions.

Instead if we split the face into Delaunay triangles, we can simply swap each triangle and in this way it will keep the proportions and also it will match the expressions of the new face, like for example if you smile, close eyes or open the mouth.

* Requirements on destination image

The triangulation of the destination image needs to have the same patterns of the triangulation of the source image. That means that the connection of the points has to be the same.

So after we do the triangulation of the source image, from that triangulation we take the indexes of the landmark points so that we can replicate the same triangulation on the destination image.

|  |  |
| --- | --- |
|  |  |
| *Delaunay triangles in source image* | *Delaunay triangles in destination image* |

***Step 3:*** Extract and warp triangles

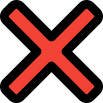
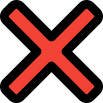
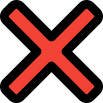
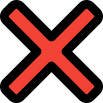
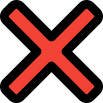
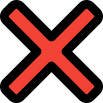
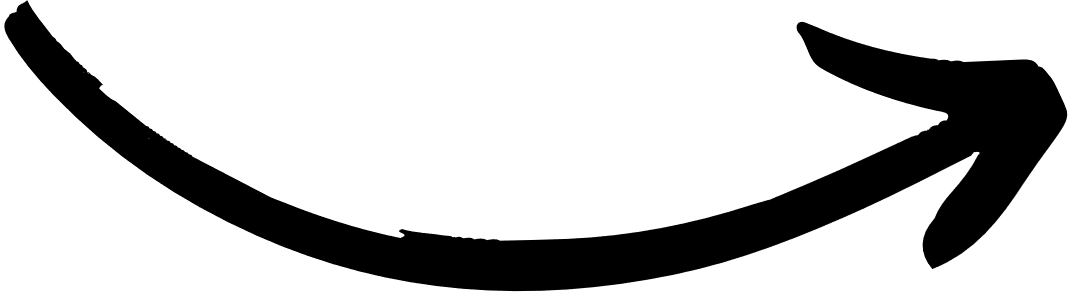
Once we have the triangulation of both faces we take the triangles of the source face and we extract them.

We also need to take the coordinates of the triangles of the destination face, so that we can warp the triangles of the source face to have same size and perspective of the matching triangle on the destination face. This step uses Affine Transformation to warp those triangles.

In Open CV library, it supports Affine Transformation through two function

M = **cv2.getAffineTransform** (origin\_triangle\_points, target\_triangle\_points)

* Input: Three pairs of corresponding points
* Output: The transformation matrix



warped\_triangle = **cv2.warpAffine** (origin\_cropped\_rectangle, M, (w,h))

* Input: The transformation matrix
* Output: An image

This function multiply each pixel in the source triangle by that matrix to get the triangle destination.

***Step 4:*** Link the warped triangles together

Once we have cutted and warped all the triangles we need to link them together.

We simply rebuild the face using the triangulation pattern, with the only difference that this time we put the warped triangle.

|  |  |
| --- | --- |
|  |  |

After warp each triangle together, we will get the face of the source image as the face of the destination image without changing the original features of the source image such as eyes, nose and mouth.

***Step 5:*** Replace the face on the destination image

The face is now ready to be replaced. We cut out the face of the destination image to make space for the new face.

So we take the new face, and the destination image without face and we link them together.



***Step 6:*** Seamless Cloning

It is easy to see that the skin color of these two people has a big difference, so we need to adjust the skin color so that the final image looks the most natural.

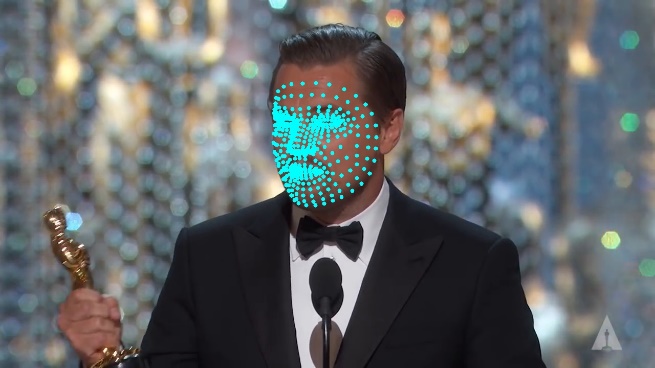
On Opencv we have a built in function called “ **cv2.seamlessClone** ” that does this operation automatically.



## 2.3 Face Swap Using OPENGL

***Step 1:*** Get the landmark points of both images

In this first phase, we deploy the pre-trained model known as “Mediapipe Face Mesh” model detect 468 3D face landmarks in both destination image and the source image.

A picture containing person, indoor, clothing, suit

Description automatically generated

*Landmarks in the source image Landmarks in the destination image*

***Step 2:*** Map the source landmarks to the available canonical face model

In the second phase, we load the canonical face object from the Internet and map all the landmarks in the source image to the object in 2D form. The method for mapping is called triangulation and warping triangles. This means we divide the face in a number of triangles gotten from “Face Mesh” model produced by ‘Mediapipe’, and each triangle of the source image are then warped in order to be fitted to every triangle in the canonical face model. After this phase, we have the finished 2D face model that has the source face in that.

A picture containing text, clothing, wearing, person

Description automatically generated

*This is called the texture of the source face in 2D coordination.*

***Step 3:*** Load the face texture and adjust the 3D mask

In this section, we will load the face texture from the previous phase and use OPENGL library to adjust the mask. First, the canonical face model object has 3 components which are ‘vertex’, ‘texture’ and ‘face’. The first component ‘vertex’ gives us the locations of the landmarks in the real image. The second provides us with the locations of those landmarks in the screen and the final component gives us all the triangles from the landmarks and their locations in the screen. At first, we copy 468 landmarks found in the destination image and paste them into the vertices of the object. We then load the face model and adjust the direction and the size of the face mask as the original object is in a wrong direction as well as its size.

***Step 4:*** Replace the face on the destination image

After having the landmarks of the destination image, we draw the convex hull of the face and make a black mask from the hull. Ultimately, we will use **cv2.seamlessClone** to put the source model face into the drawn hull in the destination image. The function then automatically alters the color of the source’s skin to fit the color of the destination’s skin.

A picture containing graphical user interface

Description automatically generated

# CHAPTER 3: RESULT & COMPARISON

## 3.1 Result

In my group's experiment, my group will swap the face of a girl on the website [*thispersondoesnotexist.com*](https://thispersondoesnotexist.com)to the face of a singing singer. For video, each frame of a video is an image so just swap out all the frames in the video based on the methods shown earlier.

|  |  |
| --- | --- |
|  |  |
| *Source image* | *Destination video* |

***Result***

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| *68\_facial\_landmark*  *Face Swap* | *81\_facial\_landmark*  *Face Swap Delaunay* | *mediapipe*  *Face Swap Delaunay* | *mediapipe + opengl*  *Face Swap Delaunay* |

## 3.2 Comparison

Based on the results, my group ranks them in order from low to high :

1. **68\_facial\_landmark Face Swap**



Due to only eyes, mouth, nose and eyebrow are swaped the face lacks expression and the recognition score is not high

1. **Mediapipe + opengl Face Swap Delaunay**



The recognition score is a little bit higher but the face also looks unnatural because every face has different depth and size

1. **81\_facial\_landmark Face Swap Delaunay**



Has 81 facial landmark points rather than 68 so the final result improved compared to the method above, points like nose and mouth blink according to mouth gesture

1. **Mediapipe Face Swap Delaunay**



Face looks clearer and more detailed

Click [*this*](https://drive.google.com/drive/folders/1Y2Q7NbDe3G0MLdxUo6Q1YU05SAVNVJr_) to see the results as well as the source code of the above four installation methods.