A Project Report On

Facial Feature Detection and Reenactment Onto Computer Generated Imagery

Submitted in partial fulfillment of the requirements for the award of the degree Of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by:

Abhishek Pande (17115005)

Arpit Gupta (17116015)

Under the guidance of:

Dr. Debashis Ghosh

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE – 247667 (INDIA)

June 2021

CANDIDATE'S DECLARATION

I hereby certify that the work presented in this report entitled "Facial Feature Detection and Reenactment Onto Computer Generated Imagery" in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology is carried out by me during the period of July 2020 to June 2021 under the supervision of Dr. **Debashis Ghosh.**

This work has not been submitted elsewhere for the award of a degree/diploma/certificate.

Date: June 4, 2021

Place: Indian Institute of Technology, Roorkee

Abhishek Pande	Arpit Gupta
……..	……..
Name of Student-1	Name of Student-2

Facial Feature Detection and Reenactment Onto Computer Generated Imagery

Abhishek Pande, 17115005, Arpit Gupta, 17116015

***Abstract*—Real-time emotion extraction and transfer are fundamental and open problems in Computer Vision and Computer Graphics. With studies ranging from analysis and creation of facial reenactment, deep-fakes to more commercial domains like creating real-life Computer Generated Imagery (CGI) in movies, with exact human-like facial emotions, the above-stated problems find applications in various diverse fields.**

This paper presents a novel method of facial reenactment of a monocular video target sequence in real-time. The source is a video sequence captured from a commercial commodity webcam. Our goal is to transfer the extracted facial features and animate them upon a Computer Generated Imagery (CGI). To this end, we address the problem of facial feature extraction from a monocular video using *dlib* and solve the head-pose estimation problem. Secondly, we propose a method to capture and transfer fine wrinkle-level details displayed in various facial expressions.

***Index Terms*—Computer Generated Imagery, dlib, PnP, Blender**

I. Introduction

REAL-time facial feature extraction and transfer have a long history in computer vision. Some fruitful approaches have relied on methods and techniques such as area localization, gaze detection, Convolution Neural Networks based emotion detection, etc. In this project, we present a way of capturing facial features and transferring them onto a CGI.

In our method, we do the proposed by first tracking the facial features using the facial landmark data and dlib Python library. The algorithm is based upon a model trained with the help of multiple images in which the facial landmark features are marked and then passed through the ensemble of regression trees. After training, these regression trees are capable of estimating the facial landmark. We propose a canny edge detection-based algorithm to obtain finer details of facial expressions like forehead wrinkles, mid-head wrinkles, etc. We also offer to calculate the number of wrinkle lines obtained and differentiating the received lines from noise obtained on running edge detector over video frames.

The facial landmarks are brought in the form of points on the face, and each point gives us the location of a prominent facial feature/landmark. We use preexisting blender functionality to transfer and animate these facial features onto the Computer Generated Imagery (CGI). This functionality allows us to animate over a CGI. We use CGI models that already have major emotion/feature animations present. Then based on the real-time data obtained from the webcam video, we trigger these animations. We also consider the moment of landmark feature points between consecutive frames of the video to determine the intensity with which the animation is triggered.

We use models that already have basic animations and can depict basic emotions in running these animations. The models we use are available online and are open-source. These models can express six basic emotions - anger, happiness, surprise, sadness, fear, and disgust. These emotions are reflected on a human face by the movement of eyebrows, eyelids, mouth, forehead, and other fine wrinkle-level details. We reenact these emotions on the CGI model by controlling and adjusting the predefined animations of mouth, eye, head, and forehead. Other emotions such as bored, shock, devious, confused, contempt, etc., can also be achieved through various permutations and a combination of the six basic emotions.

Our work focuses on improving the current capturing and transferring facial features and emotions and reenacting them onto computer-generated imagery. Computer Generated Imagery, better known as CGI, applies computer-based graphics to create art, video games, media, films, and television. At the most basic level of CGI is making still or animated visual content using computer software.

Fig. 1: Facial CGI in Movies

The proposed methodology finds its use case in:

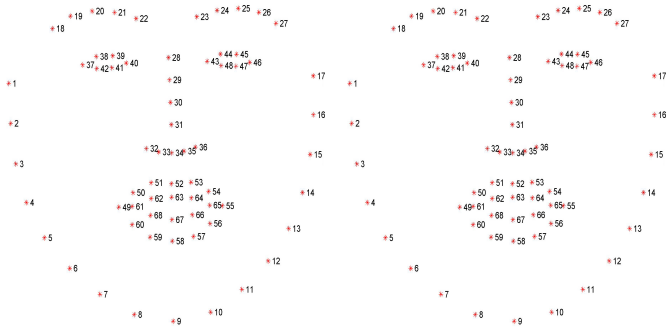
- **Animating Movie CGI, Cartoons, Video Books**: Allowing real-time facial feature tracking and transfer on-to the CGI character with fine wrinkle level details.
- **Video Reenactment**: Our proposed method can act as a stepping stone to the more comprehensive and open problem of real-time video reenactment, where we transfer the facial features from a real human face to another human face instead of CGI. This can be done by first using our method to create a mask of the first face, capturing the expressions on the mask, and then applying that mask to the target face by defining some masking algorithm compatible with ours.

II. Methodology

A. Facial Feature Detection

We use dlib to detect facial features in the incoming target video stream. We use the 68 point shape predictor data set, which maps the image of a human face onto a 128-dimensional vector space. The face recognition is then performed by mapping faces to the 128-dimensional space and then checking their euclidean distance. The accuracy in facial feature detection depends upon the distance threshold. Our method used the default threshold of 0.6, which provides an accuracy of 99.38% on the LFW facial database benchmark.

```
p = "shape_predictor_68_face_landmarks.dat"
predictor = dlib.shape_predictor(p)
detector = dlib.get_frontal_face_detector()
```



(a) (b)

Fig. 2: 68 Landmark points

The 68 points obtained are part of the **iBUG 300-W data set**, used to train the dlib facial landmark predictor. Other data sets are also available in which the number of identified points vary, e.g., 194 point model trained on **HELEN data set**. Changing the number of points affects the detailing in capturing v/s the time taken by the predictor. The predictor used identifies points on the face such as the corners of the mouth, eyebrows, eyes, and so forth.

The frontal face detector is made using the classical Histogram of Oriented Gradients (HOG) combined with a linear classifier (Support Vector Machines), image pyramid, and sliding window scheme. The predictor provides us with the method to obtain a bounding box of the face (i.e., the *(x, y*-coordinates of the face in the video stream frames. Given the face region, the algorithm begins to detect the vital facial structures in the face region. To estimate the landmark locations, the shape predictor uses the following algorithm:

- Examines sparse set of input pixel intensities(features to the input model)
- Passes the features into the Ensemble of Regression Trees(ERT)
- The predictor locations are refined to improve the accuracy through a cascade of regressors

The result is a shape predictor that functions in real-time.

We run the algorithm on the incoming video frames in real-time and obtain the 68 landmark points located in 2D coordinates.

The face detector and predictor algorithm provided by the *dlib* library used for our implementation is created using techniques such as HoG filters, linear classifiers, image pyramid, and sliding window detection schemes. The aim is to extract features into a vector using the Histogram of Oriented Gradients and feed the vector into a Support Vector Machine (Linear Classifier) which will assess whether the region contains a face or not.

1) Histogram of Oriented Gradients: Histogram of oriented gradients (HOG) is a feature descriptor used popularly used in computer vision for object detection.

As a feature descriptor, HOG helps in representing an image in simplified form by extracting useful information and discarding extraneous information. HOG takes a picture of the form $n \times m \times d$, where n, m are the height and width of the image, respectively, and d is the number of channels present in the image and returns and processed output vector. The output vector in itself does not help view purposes. Still, it is beneficial for image recognition and object detection, which is the primary function of recognizing and detecting facial features.

In HOG, the histograms (distribution) of the direction of oriented gradients are used as features. Gradients are used as their magnitude is large around edges and corners and packs more information than flat surfaces.

The magnitude and gradient are calculated using (1)

$$g = \sqrt{g_x^2 + g_y^2}$$
$$\theta = \arctan \frac{g_x}{g_y} \tag{1}$$

The processed gradient image highlights the outline in the image. After gradients are calculated, the image is divided into 8×8 cells, and histograms for the oriented gradients are calculated. The gradient of each such patch contains $8 \times 8 \times 2 = 128$ numbers which are represented using a nine bin histogram. The final step is to normalize the histogram and calculate the feature vector.

2) Support Vector Machine (Linear Classifier: Support Vector Machine is a supervised machine learning algorithm mainly used for classification problems. Each data item is plotted in the SVM algorithm as a position in n-dimensional space, with n being the number of features. The value of each feature is the value of a particular coordinate. The classification is performed by differentiating the hyper-plane into various classes.

(2) represents our model/classifier as a function.

$$h_{w,b}(x) = g\left(w^T x + b\right) \tag{2}$$

3) Image Pyramid: Image Pyramids are used to work on the image in different resolutions, generally searching objects in the picture. In these cases, we create images and search for the thing in all these images. These sets of images with

different resolutions are called *Image Pyramids*.
There are two types of pyramids possible :

- Upsize the image, i.e., Zoom in
- Downsize the image, i.e., Zoom out

Every layer of the pyramid (image in the set of images) is numbered from bottom to top and is denoted as G_i, and layer G_{i+1} is always smaller than layer G_i in an upward pyramid. We convolve G_i with a *gaussian kernel* and remove every even-numbered row and column to create a new layer.

$$\frac{1}{16}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix} \tag{3}$$

The resulting image is precisely one-quarter the area of its predecessor image.

B. Head Pose Estimation

Head pose is an orientation of the face with respect to the camera; it is the orientation of the face with respect to the screen or the camera from which the source video is obtained. The orientation/pose can be changed by either moving the camera or moving the head.
The head's movement is an important feature; our head's position affects the essential facial features/landmarks and captures. Moreover, as the aim of our project is to improve the CGI facial capture methods, a focus has to be given to capturing the movement of the head and estimate the current pose and transfer it to from the video frame (source: real human face) to the target (CGI face).

An alternate name for the problem *head pose estimation* is the *Perspective-n-Point* problem or *PNP* problem. The target of the problem is to detect the orientation of the face with respect to the camera when we have the n-3D points of the face and their corresponding 2D projection.

To solve the head pose estimation problem, we solve the Perspective n Point problem. We have used the *solvePnP* function provided by OpenCV, which implements various algorithmic approaches. The type of function to be used to solve the problem is decided by passing a flag.We have used the default flag for our method, which is essentially the *Direct Linear Transform (DLT)*. The flag is called *SOLVEPNP_ITERATIVE*. This particular algorithm also includes Levenberg-Marquardt optimization.

We first classify the landmark points that are passed onto the function to estimate the head pose. An implementation of the algorithm is explained in the pseudo code (Algorithm 1).

1) Mathematical Representation of Camera Model:

- **Translation** : the movement of points in 3D from a location (X, Y, Z) to a new location (X', Y', Z'). These movements have three degree of freedom.
- **Rotation** : rotation also provides three degree of freedom and is generally represented in terms of Euler angles.

So, translation and rotation are needed to be computed for the estimation of a 3D pose.

Algorithm 1 Head Pose Estimation

detector ← frontal face detector using dlib
predictor ← 68 landmark file to shape predictor in dlib
cap ← VideoCapture in openCV
frame_width ← frame width from cap
frame_height ← frame height from cap
fps ← fps from cap
model_points ← model point tuple
image_points ← image point tuple
for n = 1 to VideoLength **do**
    image ← capture video frame
    frame ← resize captured frame
    gray ← store image as gray scale converted image
    rects ← detector(gray, 0)
    for rect in rects **do**
        (bx, by, bw, bm) ← found the bounding box dimension using face_utils
        plot it on the original image
        shape ← predictor(gray, rect)
        Convert the shape obtained to numpy array
        for $(i, (x, y))$ in $enumerate(Shape)$ **do**
            Check if i is equal to a key landmark point
            map these values on the image
        end for
        focal_length ← focal length of the webcam
        center ← optical center of the webcam
        camera_matrix ← generate camera matrix using numpy
        (rotation_vector, translation_vector) ← solvePnP(model_points, image_points, camera_matrix, dist_coeffs, flag)
        Find the head pose estimation using slope and Jacobian using rotation and translation vector obtained
    end for
end for

2) Working of PNP Algorithm: PNP algorithm is one of the oldest algorithms for head detection. In our task, we are constrained to the 3D coordinate system. Our job is to convert the 3D world coordinates into camera coordinates. To achieve this, we use the intrinsic properties of the camera to project camera coordinates.

Let us assume (U, V, W) be the location of a 3D point in the World Coordinates; now, if we have prior knowledge of the rotation **R**, we can use it to map (X, Y, Z) in 3D coordinates to camera coordinates with the help of the translation matrix **t**.
Here, **R** is 3×3 matrix and **t** is a 3×1 matrix.

$$\begin{bmatrix}X\\ Y\\ Z\end{bmatrix} = R\begin{bmatrix}U\\ V\\ W\end{bmatrix} + t \tag{4}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [R|t] \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \tag{5}$$

We can rewrite (5) in expanded form as follows :

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{0_0} & r_{0_1} & r_{0_2} & t_x \\ r_{1_0} & r_{1_1} & r_{1_2} & t_y \\ r_{2_0} & r_{2_1} & r_{2_2} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \tag{6}$$

This provides us with multiple linear equations, which can be solved if a sufficient number of correspondence points are known.

C. Blender and Animations in Blender

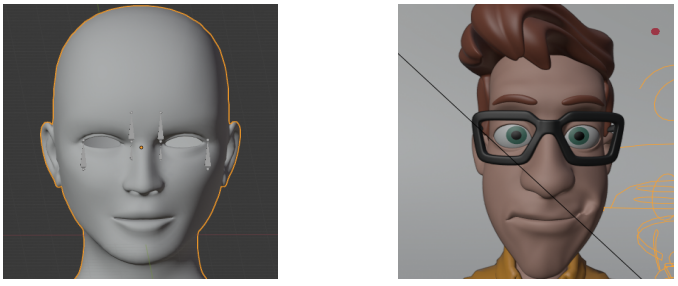
" Blender is the free and open-source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, video editing, and 2D animation pipeline." The following is the definition of Blender provided on Blender's official home page.

We have pre-designed blender models to transfer facial landmarks features on-to them. Blender allows animating static models by providing functionalities like shape-keys and motion paths. These allow deforming existing shapes(on the models) into new forms for animations and allows in to visualize the motion of points as the path over a series of frames. Each animation is called a **key-frame** and can be run by triggering the key-frame. These key-frames/animations have a value lying in a specific range generally from 0-1, denoting the intensity of animation to be run. The intensity here refers to the deformity of the shape from its original place **, e.g. the stretch of a smile on a person's face**, etc.

Another useful functionality provided by Blender is **Bones**. Bones make a modal poseable and help in shifting models. These are modeling tools that are important for animating characters.

For our work we have used two Blender model :

- **Vincent** created by Andy Goralczyk and Juan Pablo Bouza.
- **Grey-Face Model** created by Pierrick Picaut.



(a) Grey Face (b) Vincent

Fig. 3: 3D Blender Models

Vincent's model provides us with the key-frames to animate mouth, eyebrows, lips, eyes, and head movement. The Grey-Face model provides wrinkle-level details, which the Vincent model does not offer. Although the tracking and extraction functionality is independent of the CGI, the reenactment portion depends on the detailing in the used CGI.

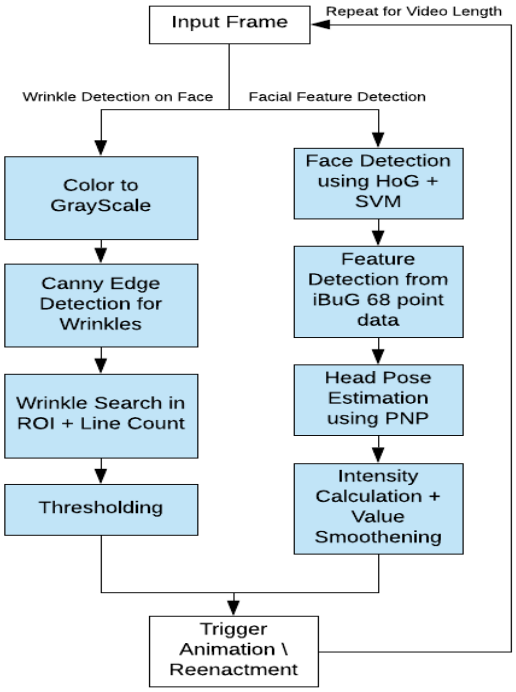


Fig. 4: Methodology Flow Chart for Feature Reenactment

D. Reenactment

Facial Feature Reenactment refers to the process of transferring/reenacting the captured features from the source (Real-Time Webcam Video Stream) to the target (Blender Mesh Model).

We have already captured facial features/landmarks by the methods explained in the above sub-sections. To transfer them onto the target, we use the key-frame properties of blender models that allow us to animate various parts of the model along a fixed motion path. The Vincent and Grey Face model that we have selected already has pre-designed animations which allow the model to reshape itself. The animations are :

1) Eyebrows animation
2) Mouth/lip animation
3) Eye + Forehead wrinkle animation
4) Mouth wrinkle animation

An implementation of the algorithm is explained in the pseudo code (Algorithm 2).

Given these animations, we execute these animations whenever we discover the corresponding facial feature. Attention has to be paid to the intensity of the animation to be performed. To solve this problem, we have developed a

Algorithm 2 Facial Feature Reenactment

for frame in frames **do**
  bones ← model's bones that have attached animations
  get_range ← intensity values for the animation
  smooth_transition ← Smoothen the value between consecutive frame transition
  bones["head"] ← pose estimation for up/down motion of head
  bones["head"] ← pose estimation for rotation of head
  bones["head"] ← pose estimation for left/right motion of head
  bones["mouth"] ← mouth movement along horizontal axis
  bones["mouth"] ← mouth movement along vertical axis
  bones["brow_L"] ← left eyebrow movement
  bones["brow_R"] ← right eyebrow movement
  bones ← execute bone animation for current frame
end for

normalizing algorithm that tracks the change in landmark points provided by *dlib*. Based on the magnitude of change in-between frames, calculates the intensity to be assigned to the animation. The human facial emotion transitions are smooth. We have also developed a smoothening algorithm that provides a smooth transition for the changing animation intensities for the Blender model.

To reenact wrinkle level details, we followed a different approach with intensity. As *dlib's 68 landmark features* do not provide points to track forehead wrinkles and a custom predictor was not created because of the lack of data set for the same, we cannot track wrinkle movement in-between consecutive frames. To solve this problem, we devised the algorithm to count wrinkle lines on various sections face like forehead, mid-head, and mouth, as explained in the following section. Suppose the number of calculated lines is more significant than a certain threshold. In that case, we run the wrinkle animation with the intensity of 1 lest we assume the value to be random noise generated by processing the image through a canny edge detector. This way, we can reenact wrinkle-level details with some accuracy.

E. Wrinkle Detection

The wrinkle detection problem can be considered as an edge or ridge detection problem. To solve this, we employ the Canny Edge Detection technique to detect wrinkles over the face better. Since the edge-detection is susceptible to noise and unwanted details, filtering it out is essential to avoid false results.

The method initially involves smoothening the image. We use a low pass filter or a first-order Gaussian filter to reduce the high-frequency components caused by the noise. Smoothened image is then convolved with a Sobel filter. As an edge can point in any direction, we compute the gradient along both x-axis (G_X) and y-axis (G_Y) using Sobel filter and round off the direction in one of the four directions: vertical, horizontal, or one of the two diagonals. We calculate the magnitude and direction of the edge using the following equations :

$$\begin{aligned} G &= \sqrt{G_X^2 + G_Y^2} \\ \theta &= tan^{-1}\left(G_y/G_x\right) \end{aligned} \tag{7}$$

Sobel filter calculates the difference between pixel intensities near the edge region and works similarly as a first-order derivative. The following is the horizontal mask of the Sobel filter(8), since zero column is in the horizontal direction:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{8}$$

To classify between the strong and weak edges, we use the concept of thresholding. We employ the hysteresis curve to obtain the thresholds ($Th1$ and $Th2$, $Th1 > Th2$) for the edges. Edges with an intensity gradient greater than $Th1$ are sure edges, and those below $Th2$ are the weak ones that are discarded. Gradients between these two values may or may not be classified as an edge based on their connectivity.

$$\begin{aligned} med\_val &= np.median(image\_gray) \\ Th1 &= int(min(255, 1.3 * med\_val)) \\ Th2 &= int(max(0, 0.7 * med\_val)) \end{aligned} \tag{9}$$

The above equations can be used for the Hysteresis Thresholding to find the upper and lower value of the thresholds. An implementation of the wrinkle detection algorithm is explained in the pseudo code (Algorithm 3).

Algorithm 3 Wrinkle detection

cap ← VideoCapture in openCV
frame_width ← frame width from cap
frame_height ← frame height from cap
fps ← fps from cap
for n = 1 to VideoLength **do**
  image ← capture video frame
  gray ← store image as gray scale converted image
  gaussian ← Apply Gaussian filter on the gray image
  median ← median value of pixels of gaussian image
  upper ← upper threshold value as per equation 9
  lower ← lower threshold value as per equation 10
  canny ← Canny edge detection on gaussian image
  wrinkle lines ← count wrinkles from gaussian image
end for

1) Forehead Wrinkle Extraction: We fix the size of the rectangle (a one-third topmost portion of the face) for extracting the wrinkles on the forehead. It assists in diminishing the number of pixels to apply the image processing algorithms. In the beginning, we convert the image to gray-scale and extract the portion of the image in our fixed bounding area. Then, we apply Canny Edge Detection to obtain the wrinkles in the forehead area of the image. After getting the exact coordinates of the edge pixels, we count the number of

lines in the forehead. We discard all the non-horizontal lines as noise, considering all the wrinkles in the forehead are almost horizontal. We run sixteen different vertical lines in the forehead and extract the number of pixels corresponding to an edge in every line; let's call those pixels ep. To compute the number of wrinkles in the forehead, we use the following equation:

$$ForeheadWrinkles = (ep_{total}/ep_{max}) \quad (10)$$

2) Mid-Forehead Wrinkle Extraction: The methodology for extracting the wrinkles in the mid-forehead region is almost similar to the forehead wrinkle extraction. We covert the image to gray-scale and select the area in the fixed rectangular bounding box of size 20x28 pixels in the region between the eyebrows. We apply the Canny Edge Detection technique to identify the edge pixels in this region. We discard all the edge pixels that are non-vertical. We use seven horizontal lines and use a similar algorithm to calculate the number of wrinkles.

III. Results and Discussion

Changing the number of points from 68 to other values affects the detailing in capturing v/s the time taken by the predictor. Intrinsic parameters of the camera, such as optical center, are calculated by the center of the image and the focal length by the width of the image in pixels. Head Pose estimation requires the location of 6 landmarks, namely the tip of the nose, left corner of the left eye, chin, right corner of the right eye, left corner of the mouth, right corner of the mouth.

The end result is a shape predictor that functions in real-time. We run the algorithm on the incoming video frames in real-time and obtain the location of the 68 landmark points in the form of 2D coordinates and the head pose estimation. We are using an RGB image and applying the discussed algorithms.

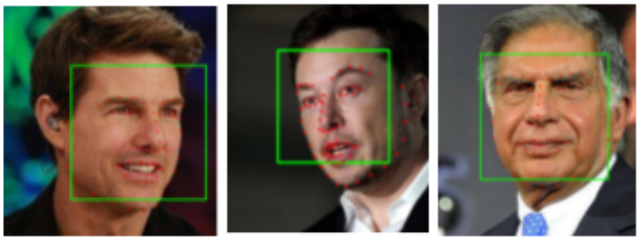


Fig. 5: Detection of 68 facial landmarks

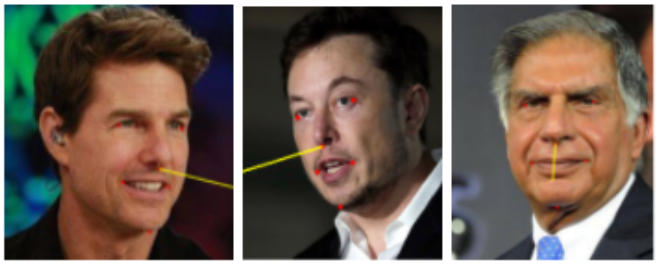


Fig. 6: Head Pose Estimation

The 68 facial landmarks and the head pose estimation are then calculated in real-time using blender technology. They are enacted on to the CGI in real-time using the blender technology. The Vincent model is used as shown, which provides us with key-frames to animate the six landmarks.

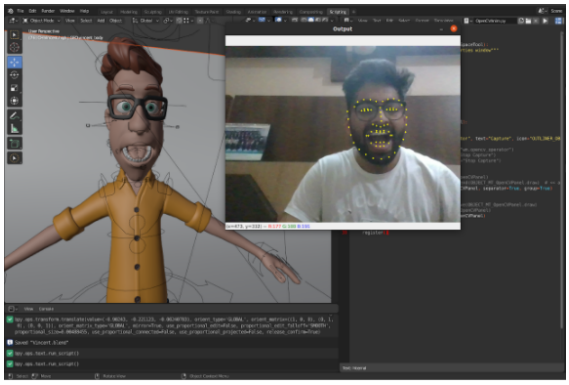


Fig. 7: Blender Animation (Vincent Model) Pose 1

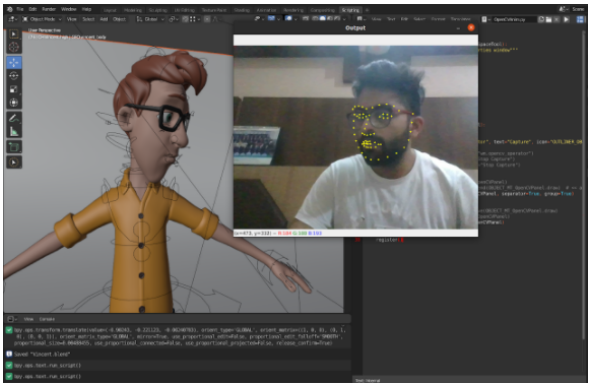


Fig. 8: Blender Animation (Vincent Model) Pose 2

The Canny edge detection technique was used for the detection of wrinkles. We pass a low pass Gaussian filter of dimensions 5*5 over the image to filter out the undesired noise components, on which the canny edge detection was applied. Then the algorithm described in the above section is used to count the number of wrinkles on the forehead and mid-forehead. The resulting pictures are shown below as:

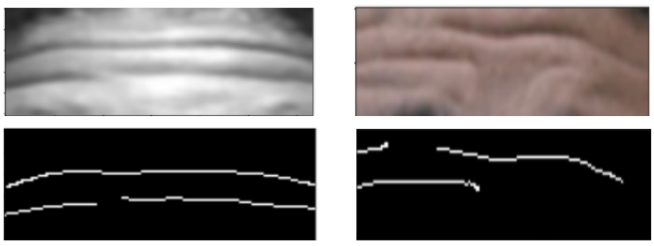


Fig. 9: Wrinkle detection - 3 major wrinkles

IV. Conclusion

The presently developed approaches perform decently in terms of the computational costs as it uses efficient and robust

algorithms. The dlib shape predictor does not provide us with wrinkle level details, nor does it help estimate the head pose. The PNP approach can further optimize by not computing the head pose for every frame. This optimization will lead us to a trade-off between execution speed and accuracy. Reenactment of the complete facial details will be done on the CGI using Blender functionality.

Computing a complete model on Blender again performs well in terms of computation. This technique can be used to reenact facial features on a CGI in real-time, as evident by the results. However, the more we try to capture the finer details like the wrinkles, eye-gazing, etc., the more time it takes for the model to reenact on the Blender. Capturing more delicate information would increase the accuracy, but it would require a sound hardware system to perform such tasks. Robust algorithms are necessary to overcome this problem.

V. Limitations

The method proposed works in reenacting facial emotions with wrinkle/finer level details to CGI in real-time. There are still some limitations to the proposed techniques :

- **GPU Support** : The proposed algorithm works well on a standard available system with RAM around 2-4 GB and possible (not necessary) GPU support. But the computational power required to reenact the emotions/features onto the 3D model is quite heavy. The reason for this is re-rendering the 3D model's pixels/vectors for each frame of the source video.
  The method does not have GPU support as any of its dependency and will function on any standard system, but there is a substantial lag witnessed. A better processing machine might reduce the lag for running the method. The above one is a hardware-based solution.
  A technique-based solution for this could be skipping some frames, i.e., not reenacting every frame. An algorithm can be included which will track the facial features between consecutive frames and only reenact them onto the 3D model when there is a substantial change in facial features.
- **3D Mesh Model** : Blender plays a very crucial role in the proposed methodology. The tracking algorithm for facial features and wrinkles is independent of the 3D model, but the reenactment depends heavily on the kind of 3D mesh model used. Models with an exemplary level of detailing will represent human emotions with greater accuracy and realism.
  To overcome this limitation, a solution is proposed in the **Future Prospects** section, suggesting a dynamic mesh creation based on the facial feature and is computationally heavy.
- **Age factor** : As we have used the standard 68 points landmark data set based on iBUG 300-W data. Hence, we detect wrinkles by considering them as edges on the otherwise smooth skin using a canny edge detector. By counting the wrinkle lines, we noticed we have a threshold on whether to trigger the animation on the 3D model.
  This method has a basic assumption that a person will not have any wrinkle when not depicting any emotion, which is not valid for older adults where wrinkles are a natural feature with passing age. Our algorithm will count these wrinkles and trigger the animation irrespective.

VI. Future Prospects

The presently developed approaches perform decently in terms of the computational costs as it uses efficient and robust image processing algorithms. We have explored various pros and cons of this approach we have used in most of our steps. We mention some of our findings that could improve our system as a whole.

A. Adaptive Divisioning

The resolution of the mesh is relatively low for capturing the various details of the input image using the vertex displacement methodology. So, we will use a hierarchical reconstruction methodology to capture more delicate and subtle details of the input image. We will start with the coarse mesh and will iteratively subdivide it into finer meshes. Thus, we will generate an image pyramid with the lowest-resolution mesh at the top and the highest-resolution mesh at the bottom.

We can use 4-8 subdivisions to attain a proper hierarchy and divide the mesh adaptively. We prefer this methodology as it does not lead to any edge breaks and generates relatively simple divisions. Also, our mesh is a triangulated quadrilateral, which is a perfect fit for this division methodology.

We can visualize the subdivision technique from figure X. The triangle $t1$ has two vertices $v1$ and $v3$ with valency eight while one vertex $v2$ with valency four. The development of subdivisions is done by bisecting the edge $v1v3$ to create a new vertex $v4$ and constructing the two new edges to connect $v4$ with the two vertices valency of four. We use two masks, the face mask and the coordinate mask, to obtain the new vertices.

We can construct the $(i+1)^{th}$ mesh from the i^{th} mesh by iterating over all the faces with the adaptMesh function. The algorithm terminates when the interior edge's length goes below a particular threshold value in the image space. Also, the adjacent triangles in the quadrilaterals must have a similar subdivision level to avoid cracks. We use the adaptCoordinate function to update the positions of the new vertices that are obtained. The hierarchical mesh is created only once at the start of the video, while we update and refine the mesh during runtime using the same face and coordinate masks.

B. Gaze Detection and Eyelid tracking

Eyes are a notable facial characteristic, and as future work on the algorithm, we can include eye gaze detection and reenactment. The process of reenactment remains the same. Although dlib in itself tracks the boundaries of the eyes, the preexisting algorithm can be enhanced.
The first step to this would be pupil(center of the eye) detection in the human eye. For this, we can use isophote curvatures

Algorithm 4 Adaptive Subdivision

function $adaptMesh(V, T, t)$
  if *t has not been processed* **then**
    $edge \leftarrow findInteriorEdge(t)$
    if *projection(edge)* $> \epsilon$ **then**
      while $t.neighbour.level < t.level$ **do**
        $adaptMesh(V, T, t.neighbour)$
      end while
      $adaptVertex(V, T, edge.start, t.level)$
      $adaptVertex(V, T, edge.end, t.level)$
      $v \leftarrow bisectEdge(edge)$
      $insert\ v\ in\ V$
      $t1 \leftarrow bisectTriangle(t)$
      $t2 \leftarrow bisectTriangle(t.neighbour)$
      $insert\ t1, t2\ in\ T$
    end if
  end if
end function
function $adaptCoordinate(V, T, v, l)$
  if *v has not been processed* **then**
    for t in T containing v **do**
      while $t.level < l$ **do**
        $adaptMesh(V, T, t)$
      end while
    end for
    $updateVertexPosition(v)$
  end if
end function

whose sign is influenced by the intensity of the curve's outer edge. We can infer that center of the eye is the maximum isocenter and is the local maxima in the curve.

C. Better Facial Tracking in Cases of Extreme Head Movements

The algorithm works great for fixed head pose and slight movements of the head. We have observed that the accuracy reduces when the head poses changes abruptly.

The facial features are not appropriately tracked in these situations, and the reenactment is not done correctly. Therefore to detect head pose changes beyond speed and angles, we can use some pre-trained data to detect the head pose.

REFERENCES

[1] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1867-1874, doi: 10.1109/CVPR.2014.241.

[2] B. A. Efraty, M. Papadakis, A. Profitt, S. Shah and I. A. Kakadiaris, "Facial component-landmark detection," *2011 IEEE International Conference on Automatic Face and Gesture Recognition (FG)*, 2011, pp. 278-285, doi: 10.1109/FG.2011.5771411.

[3] T. Vatahska, M. Bennewitz and S. Behnke, "Feature-based head pose estimation from images," *2007 7th IEEE-RAS International Conference on Humanoid Robots*, 2007, pp. 330-335, doi: 10.1109/ICHR.2007.4813889.

[4] Q. Wang and L. Shi, "Pose estimation based on PnP algorithm for the racket of table tennis robot," *2013 25th Chinese Control and Decision Conference (CCDC)*, 2013, pp. 2642-2647, doi: 10.1109/CCDC.2013.6561387.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

[6] B. Wang and S. Fan, "An Improved CANNY Edge Detection Algorithm," *2009 Second International Workshop on Computer Science and Engineering*, 2009, pp. 497-500, doi: 10.1109/WCSE.2009.718.

[7] J. Canny, "A Computational Approach to Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[8] N. Boyko, O. Basystiuk and N. Shakhovska, "Performance Evaluation and Comparison of Software for Face Recognition, Based on Dlib and Opencv Library," *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, 2018, pp. 478-482, doi: 10.1109/DSMP.2018.8478556.

[9] Luming Ma and Zhigang Deng. 2019. Real-time hierarchical facial performance capture. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '19)*. Association for Computing Machinery, New York, NY, USA, Article 11, 1–10. DOI:https://doi.org/10.1145/3306131.3317016