3D face reconstruction from 3 images

Manish Kumar Mishra, *m.k.mishra@student.utwente.nl, s2801620, MSc. Data Science*
Nischal Neupane, *n.neupane@student.utwente.nl, s2801620, MSc. Data Science*
and Peshmerge Morad, *p.morad@student.utwente.nl, s2801620, MSc. Data Science*

***Abstract*—3D face Reconstruction is a widely researched topic in Image Processing today. However, its applicability spans various fields, from face recognition technology used in security systems and smartphones to medical applications for grading facial paralysis due to neurological disorders. In this paper, we explore the possibility of using three images of subjects from different points of view to reconstruct a 3D face. We discuss the employed methods along with their results and their limitations.**

***Keywords*—Camera Calibration, Stereo Vision, Image Processing, 3D Reconstruction**

I. Introduction

THE field of 3D (face) reconstruction has been an active research field in computer vision and computer graphics for decades. The 3D face reconstruction is the process of extracting the shape and appearances of faces from two or more 2D face images. The applications of 3D face reconstruction span from plastic surgery simulations, facial recognition in security systems, and Augmented Reality/Virtual Reality games, to even recognizing facial paralysis.

Throughout the past decade, there have been vast improvements in technology and methodology for 3D face recognition. In cinematography alone, there have been substantial improvements in computer vision. The article by Makowski et al. [1] discusses the calibration of stereoscopic cameras and analysis of the 3D depth resulting in the estimation of the disparity map acquired using the video streams from professional cameras present in the film set. As Makowski et al. describe in the article, the algorithms selected for processing these video streams do not require knowledge of intrinsic and extrinsic camera parameters, thus furthering and simplifying the use of computer vision for 3D reconstruction.

Various other approaches have surfaced for 3D face recognition. The paper by Bolkart et al. [2] discusses the use of a single image to create a geometric detail model and generate a UV displacement map, then using a regressor to predict subject-specific facial and lighting details from the image using the UV displacement map. Whereas another paper by Choi et al. [3] discusses the use of stereo images to detect 3D facial features similar to this paper. However, Choi's method involves deforming the face surface using the Thin-Plate Spline method, which will not be discussed, nor will it be used in this paper.

Although the methods discussed thus far are widely adopted and quite robust due to their application with a reconstruction of the model using a video or a single image, there are key benefits to using multiple stereo images taken from a different perspective. This paper utilizes the images taken from 3 stereo cameras from left-middle-right perspectives to reconstruct a 3D model of the face of the given subject. The use of stereo-matching will be discussed in the upcoming sections, and the exact methods which are taken to do multi-camera stereo matching and calibration to process the images leading to the outcome of a reconstructed 3D facial model.

The remaining sections of the report are organized in the following way, the used materials and the followed methodology in approaching the 3D reconstruction task are discussed in Section II. The result of our approach will be presented in Section III. Finally, our conclusions on the obtained results and shortcomings will be presented in Section IV.

II. Methods and Materials

A. Materials

For this project, we use MATLAB version R2022a with **Image Processing and Computer Vision** toolbox. This toolbox contains **Stereo Calibration Application** which is used to find the camera pair calibration parameters. The inbuilt K-means algorithm from MATLAB is also used for clustering the colour channels to isolate the face from the background.

B. Method / Implementation

1) Analysis: The challenge we want to tackle in this paper is reconstructing a 3D face mesh from multiple 2D images. In our case, we have three subjects, and for each subject, there are multiple pictures taken by the same camera setup. The camera setup consists of three cameras placed at slightly different angles. Fig. 1 shows an example pose of subject 1 taken from the three-camera setup. To achieve the goal of merging the 2D images into one 3D face mesh, we have identified the following sub-tasks. Camera calibration parameters must be extracted as the primary step in any stereo vision problem. We also need to isolate the face from the background in the images for our case. Next, we rectify image pairs according to the camera calibration parameters. Further on, we need to create disparity maps using the corrected images. We then use the disparity

Fig. 1: Same pose for Subject 1 from three cameras at different angles (Left, Middle, Right)

maps to generate the point cloud and construct the 3D face surface mesh.
Since we have three images per subject, we have decided to use two stereo pairs for the task. The first pair is the Middle-Left image and the second pair is the Middle-Right image for any scene. The subsequent paragraphs describe each of these steps in detail.

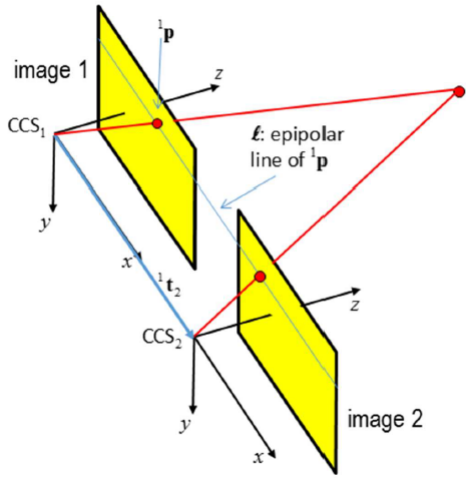

Fig. 2: Fully aligned epipolar vision [4]

2) Camera calibration: Camera calibration is the first step in any stereo vision application (binocular or trinocular). In the camera calibration step, we estimate the projection matrix, which can be decomposed into two matrices. This decomposition results in the intrinsic and extrinsic parameters for a camera pair. On the one hand, the intrinsic parameters represent the internal camera parameters, such as the camera's optical center and focal length. They allow the mapping between the camera and the pixel coordinates in the image frame. On the other hand, the extrinsic parameters define the camera's location and orientation with respect to the world coordinates frame(3D scene). The calibration is done by using images of a known object; in our case, we use checkerboard images. The squares on the checkerboard have a size of 10x10 mm^2. The University of Twente provided those images.

The provided calibration images are divided into three groups based on the camera's location with respect to the checkerboard. The locations are left, middle, and right. We have taken the center pictures as a reference point, and we have calibrated twice for middle-left and middle-right camera pairs calibration.
The calibration is done using the Stereo Camera Calibrator App from the Computer Vision Toolbox of MATLAB. The app provides a graphical user interface for performing the calibration. In addition, it enables us to review the re-projection errors in the calibration process and remove outlier image pairs to achieve the lowest overall mean error. The app helps us verify whether the calibration is done accurately by examining the rectification. The calibration image pairs must become undistorted and raw aligned if the calibration is done accurately. The Fig. 3 and Fig. 4 demonstrate the camera-centric view of the camera calibration.

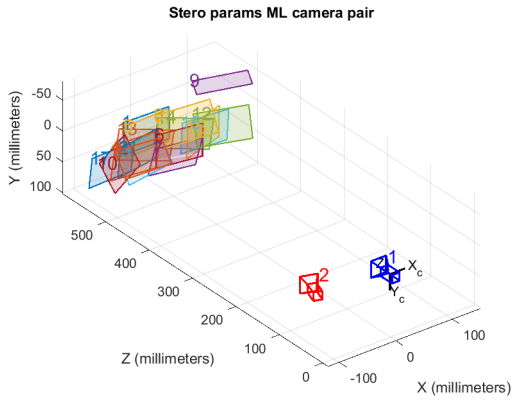

Fig. 3: Camera-centric view of the Middle-Left calibration image pairs

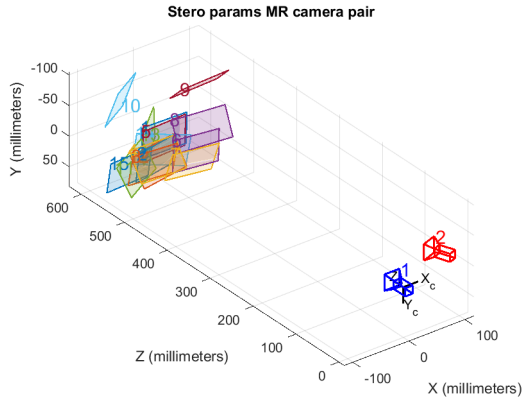

Fig. 4: Camera-centric view of the Middle-Right calibration image pairs

3) Face extraction (Background removal): The images provided for this 3D face reconstruction task are taken in a homogeneous camera setting. Each subject wears the same clothes, and the pictures are taken from three different camera angles (left, middle, right). In addition, the background color in all images is the same. We are only interested in extracting the face from all the images for this project. There are two approaches to implementing face extraction from the given images. The first approach involves detecting the human face in a given image using vision. `CascadeObjectDetector` from the Computer Vision Toolbox in MATLAB. The second approach involves using the K-means algorithm to segment the colors of the image into three clusters: background color, skin color, and clothes color. We have developed a K-means clustering algorithm based on the native K-means algorithm in MATLAB. An example provided by MATLAB heavily inspires

the developed algorithm, "**Color-Based Segmentation Using K-Means Clustering**" [5].
The first step in the algorithm is converting the image from the RGB color space to the CIELAB color space (L*a*b*). The conversion makes it easier to segment regions in the image by color without being affected by the lighting component present in the image. The L*a*b* color space is consistent with the human visual perception. The L* component represents the Luminosity component (perceptual lightness). The a* represents the Chromaticity component where the axis is relative to the green-red opponent colors, with negative values toward green and positive values toward red. The b* represents the blue-yellow opponents with positive numbers toward yellow and negative numbers towards blue. The a* and b* layers represent all the color information for the human vision (red, green, blue, and yellow). The algorithm and all its steps are demonstrated in Algorithm 1 The Fig. 5 shows the input and the output of the Face extraction algorithm. The right image shows the face extracted from the image after applying the binary mask returned from the Face extraction K-means.

Algorithm 1 Face extraction using K-Means

Require: Input Image
Ensure: Return a face mask corresponding to the face in the input image
1: Convert the image from RGB to L*a*b* color space
2: Run the MATLAB internal K-means algorithm with 3 clusters on the converted pixel values
3: Label the pixel in the original image using the K-means results
4: Create multiple images from the input image based on the color segmentation
5: Take the image where the face is located into a separate image
6: Convert the face image into a binary mask
7: Apply morphological transformations (Erosion, opening and closing) on the binary mask to ensure all face landmarks are included.
8: Return the face image binary mask

Fig. 5: Subject 1 Face (Left: the original image, Right: The extracted face after applying the binary mask from the K-means)

4) Image Pair Stereo Rectification: Following background removal using K-means clustering, stereo rectification is performed regarding the chosen subject. Stereo rectification is the process of performing a homographic transformation to cause the epipolar line of the middle image to coincide with the epipolar line of the left image, where the resulting epipolar line becomes parallel to the x-axis of the image. The exact process is also applied to the middle-right pair. Since the provided stereo image pairs are not fully aligned due to the camera placement, the epipolar lines do not coincide with each other either. This causes the problem of stereo matching to be a 2-dimensional search problem. Thus, to simplify the process and simplify this step, we transform the images (using projective transform) to align both the images. This allows for the epipolar lines to coincide, thus, making the problem a 1-dimensional search problem.

Fig. 6: Rectified image pairs (Left: Middle-Left, Right: Middle-Right) of Subject 1

In our case, stereo matching is done using MATLAB function `rectifyStereoImages`. To perform stereo rectification with the function, we need to pass the camera parameters acquired from Section II-B2 to the function. The camera parameters, representing the middle-left calibration (Fig. 3) and middle-right calibration (Fig. 4), with their respective full middle-left and middle-right images are also passed on to the function. Thus, the resulting rectified images appear as if they were shot from parallel cameras. This can be seen in Fig. 6.

Fig. 7: Rectified mask pairs (Left: Middle-Left, Right: Middle-Right) of Subject 1

Similarly, rectification is also applied to the masks acquired from removing the background. We do this because we want to use the masks in the upcoming steps (creation of disparity

maps) to extract only the facial structure avoiding the background. This results in a similar process to rectify the masks. The resulting images are shown in Fig. 7.

5) Disparity Map: The term *disparity*, in the context of vision and optics, refers to the positions of the corresponding features seen by the left and right eyes and the 2D vector between them that defines the depth in the image. However, to find the disparity map, the first step involves the "estimation of the disparities called stereo matching" [4]. As Stereo matching is the first step to computing the disparity, we explored various algorithms present to do stereo matching. However, in MATLAB, stereo matching becomes quite simple as it is performed automatically during the calculation of the disparity map. Thus, to verify the quality of the rectification process, we were able to use `Registration Estimator` application from the MATLAB toolbox. The application uses Maximally Stable External Regions (MSER) to optimize the location and the number of points, which allows for a comparison between the corresponding points to identify the quality of the rectified images.

After gauging the quality of our rectification and optimizing it as far as we could, given our knowledge of the topic, the next step involves creating the disparity map. Since our images are rectified, they are in the same range on the 2D plane, and the disparity map is solely based on the difference in the domain of x. Thus, using the Euclidean distance between the pixels, and assuming the coordinates of pixels on the left image be represented as (x_R, y_R) and the right image be represented as (x_L, y_L) and we know that rectified images have $y_L = y_R$ for every pixel, we have...

$$\sum_{x=1}^{m}\sum_{y=1}^{n} euc\_d(x) = \sqrt{(x_R - x_L)^2 + (y_R - y_L)^2}$$
$$\sum_{x=1}^{m}\sum_{y=1}^{n} euc\_d(x) = \sqrt{(x_R - x_L)^2 + 0}$$
$$\sum_{x=1}^{m} euc\_d(x) = \sqrt{(x_R - x_L)^2}$$
$$\sum_{x=1}^{m} euc\_d(x) = x_R - x_L \quad (1)$$

As shown above, the disparity for a pixel, (x, y) can be defined by simply subtracting the pixel values along the x-axis. However, this is assuming that each pixel has a corresponding point in both the rectified images. The method of calculating disparity is then performed for all the grayscale pixels in the image, thus, resulting in the creation of a disparity map.

In MATLAB, there is a built-in function for creating disparity map, namely `disparity`. This function has several options that go along with it. To make the process more robust for different subjects, we create a separate function that takes the subject, the left-middle, and the middle-right image pairs, along with the mask created after the previous step of removing the background. The first step of creating a disparity map involves converting the RGB image to a grayscale image. This is a required step as creating a disparity map of RGB image requires the separation of the color channels and creating a disparity map of individual color channels. However, this is not within the scope of this project; thus, we use a grayscale image to create the disparity map.

Algorithm 2 Disparity Map

Require: Subject, Image One, Image Two, Mask
Ensure: Return a disparity map and unreliable disparity map
1: Convert both the images from RGB to grayscale images
2: Set proper disparity range given the subject
3: Call the built-in `disparity` function with argument values specific to a subject for *'DisparityRange'*, *'UniquenessThreshold'*, and *'DistanceThreshold'*.
4: Remove the background part of the disparity map by using the provided mask
5: Smooth the disparity map by using `medfilt2` function with threshold defined based on the subject
6: Fill any holes/gaps present in the disparity map
7: Generate an unreliable values of disparity map by creating a mask for values that are extremely low and remove the unreliable disparity

Algorithm 2 describes the algorithm used to create the disparity map for the given subject as well as find and remove any unreliable disparities present in the disparity map. The disparity range used in the algorithm is another crucial aspect of the disparity map, as the range defines the depth concerning each feature of the image. For instance, the range for the nose is lower as it is closer to the camera than the range for the neck, which is further away from the camera. The range was estimated through trial and error to best represent the needed information in the disparity map based on each pixel's maximum and minimum values.

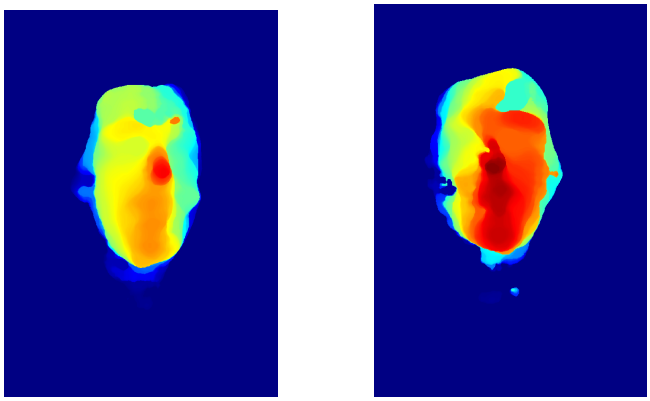

Fig. 8: Disparity map pairs (Left: Middle-Left, Right: Middle-Right) of Subject 1

The disparity map created contained the background as we used the original rectified image to create the disparity map. Thus, we had to use the mask generated from the removal of the background to remove the background from the disparity map as well. However, this leads to a ragged disparity map

that may also contain holes. Thus, median filtering was used for smoothing the disparity maps, and the holes were filled using the `imfill` function built in Matlab. The final step of creating a full disparity map involved the removal of unreliable values of the disparity map to keep the values within a reasonable range, as some values were infinitesimal, which caused a considerable scale imbalance. After the removal of the unreliable disparity values from the disparity map, the resulting images are shown in Fig. 8.

6) Point cloud and 3D mesh: Once we have the disparity map between the image pairs, we construct the point cloud by using the function `reconstructScene`. This function takes in the disparity map and the projection matrix, which is the camera calibration parameters obtained in Section II-B2. It returns the 3D world coordinates of the pixels from these inputs. An example point cloud can be found in Fig. 9. These point clouds could be merged using the function `pcmerge`. To do this, we first de-noise and downsample the point clouds and then use the ICP algorithm (MATLAB function `pcregistericp`) - which transforms and registers one point cloud to the other. However, we skip this step in the final iteration of our project.

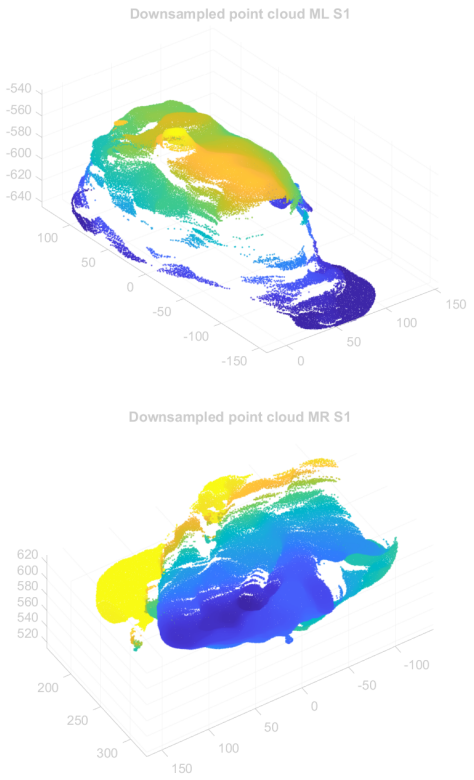

Fig. 9: (Top) Left-Middle, (Bottom) Right-Middle Point Clouds of Subject 1

The point clouds obtained, along with the disparity maps, can then be used to generate the 3D mesh of the extracted face. We utilize the sample code from [4] for creating a 3D surface mesh. In brief, the function starts by creating a set of connected triangles (mesh faces) that will form the final mesh. The function `delaunay` is used for this purpose, which takes in the disparity map values and provides the set of triangles. We also clean up the output set by removing the triangles which use the unreliable points from the disparity map. With this set of triangles, we then use the function `triangulation` to get the final 3D mesh object. Furthermore, the colors from the original image are added to make the mesh appropriately coloured as our subject's face. Sample 3D meshes can be seen in Section III.

TABLE I: RMSE obtained per pose while merging ML and MR point clouds

Subject	Pose 1	Pose 3
Subject 1	22.3207	19.4793
Subject 2	22.7217	13.2581
Subject 4	27.002	26.2866

C. Performance evaluation

Due to the time constraint, we were unable to further our project to include experiments to evaluate the performance of our project. However, we present the RMSE values from the step of point cloud merging (Section II-B6) that we had attempted in I. For the sake of this project, we have performed a manual visual inspection on the final 3D meshes obtained, as seen in Section III. We have also presented some findings and future improvement scope in Section IV.

III. Results

Some outputs from our project can be seen in Fig. 10 to Fig. 15.

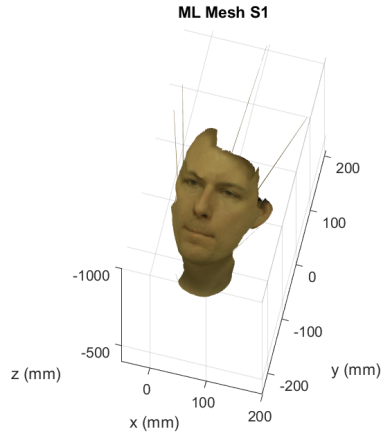

Fig. 10: Middle-Left 3D Face mesh of Subject 1

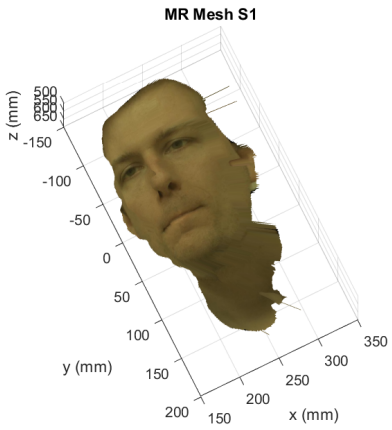

Fig. 11: Middle-Right 3D Face mesh of Subject 1

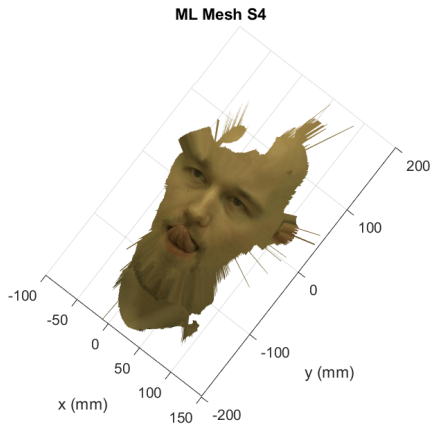

Fig. 14: Middle-Left 3D Face mesh of Subject 4

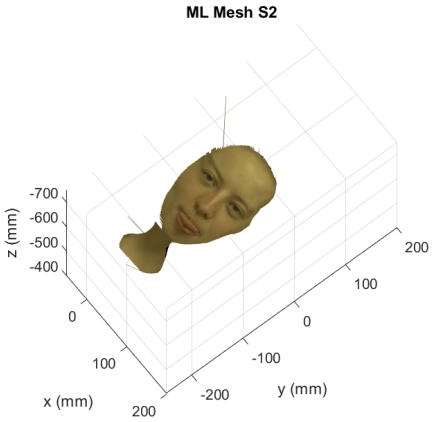

Fig. 12: Middle-Left 3D Face mesh of Subject 2

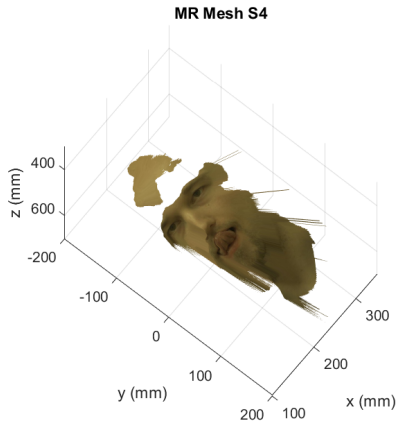

Fig. 15: Middle-Right 3D Face mesh of Subject 4

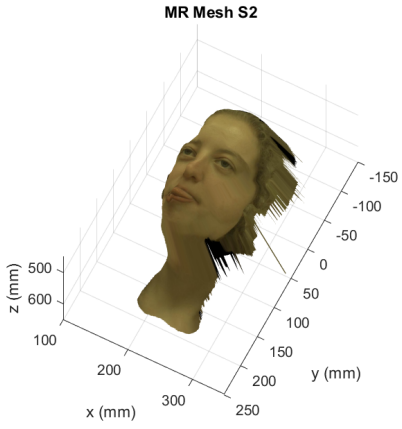

Fig. 13: Middle-Right 3D Face mesh of Subject 2

IV. Discussion and Conclusion

In our implementation, we convert the RGB image into **CIELAB** (L*a*b*) color space and only use the a* and b* channels, which contain all the required color information (as mentioned in Section II-B3). This completely removes any need for further rectifying the images for a constant brightness assumption, as we only deal with the image colors, not the brightness.

From the images in Section III, we observe some interesting findings. The face regions such as the chin and forehead seem to fail to provide the correct information for the disparity map. We can see in Fig. 15 and Fig. 14, for subject 4 that the top area of the final 3D mesh is very irregular and mostly missing from the disparity maps. This could be because there are not many different pixels in terms of color or brightness in these regions. In addition, the developed algorithm for extracting the face (as mentioned in Section II-B3) results in face regions without the hair for subject 1 and very little hair for subject 2.

In this paper, we have proposed the solution where most of the parameters used for the MATLAB functions, such as creating disparity maps and the K-means method for face detection are highly dependent on the input images. In other words, this solution may not work properly if we try to apply the developed algorithm for different subjects and thus need to be updated based on the input. A good improvement to this would be to create a more generalised application that could take in any input image pairs for a two/three-camera setup and generate the 3D mesh of the face. We believe that such an improvement will be more robust and deliver better results regardless of the nature of the supplied images.

REFERENCES

[1] P. Perek, A. Mielczarek, and D. Makowski, "High-Performance image acquisition and processing for stereoscopic diagnostic systems with the application of graphical processing units," *Sensors (Basel)*, vol. 22, no. 2, Jan. 2022.

[2] Y. Feng, H. Feng, M. J. Black, and T. Bolkart, "Learning an animatable detailed 3D face model from in-the-wild images," *ACM Transactions on Graphics (ToG), Proc. SIGGRAPH*, vol. 40, no. 4, pp. 88:1–88:13, Aug. 2021.

[3] J. Choi, G. Medioni, Y. Lin, L. Silva, O. Regina, M. Pamplona, and T. C. Faltemier, "3d face reconstruction using a single or multiple views," in *2010 20th International Conference on Pattern Recognition*, 2010, pp. 3959–3962.

[4] F. v. d. Heijden, "Stereo: from pixels to 3d surface mesh," Jan 2016.

[5] MathWorks. Color-based segmentation using k-means clustering. [Online]. Available: https://nl.mathworks.com/help/images/color-based-segmentation-using-k-means-clustering.html