Physics-based 3D Face Reconstruction

Eunsom Jeon, Hershel Millman, Mark Kapron

***Abstract* - 3D face reconstruction from 2D images is a longstanding problem in computer vision and computer graphics. Generating a 3D model from a single 2D image is an ill-posed problem because the density of 3D spatial information within an image is necessarily lower than the corresponding information in a 3D model. 3D reconstruction methods using Convolutional Neural Networks (CNN) have produced impressive results for both volumetric and surface mesh representations. Current models use feature-based learning methods and loss functions. While features are often informative, the performance of 3D reconstruction algorithms may be augmented by taking a physics-based approach.**

In this work, we present a physics-based 3D reconstruction model using CNN to create accurate facial features and geometry, considering illumination and shadow. We adopt a mesh deformation model with a physically accurate loss function.

***Index Terms* - 3D face reconstruction, physics based reconstruction, mesh deformation, differentiable rendering**

I. Introduction

3D face reconstruction is a hot topic in computer vision. In recent decades, face alignment and geometric methods have been used for detecting facial points to assist 3D face recognition for applications ranging from immersive video games to unlocking a cell phone. However, creating a 3D model from a 2D input is an ill-posed problem. A single 2D image contains information about relative sizes and colors of features from a single perspective, but a minimum of two viewpoints are required for robust analytical 3D reconstruction. Unfortunately, a stereo system for this purpose must be highly calibrated, and requires dedicated hardware. Recent studies have aimed to increase availability of 3D reconstruction by utilizing neural networks to predict 3D information given one or multiple 2D images. Convolutional Neural Networks (CNNs) are common tools for estimation of depth and 3D model parameters..

Current models for 3D reconstruction often use perceptual or feature-based loss functions to predict depth, dense aligned face coordinates, 3D Morphable Model (3DMM) coefficients, or other 3D spatial information [1]. However, previous methods of inference do not use any direct physics-based models. Thus, even though the 3D output can have a realistic geometry and structure, the output may have other attributes that are not realistic. Some of these incorrect attributes include erroneous texture and color, or unrealistic shadows and reflections. Incorporating a physics-based model should help to relieve some of these inaccuracies.

The goal of this work is to create a model which uses both deep learning and the physics of light for inference. We propose a pipeline of Convolutional Neural Networks which takes in a single image of a face, and outputs parameters of a 3D mesh.

Current techniques do not explicitly incorporate the physics of light into the algorithms. In order to improve upon these techniques, we introduce a method to reconstruct 3D face shape utilizing physically realistic differentiable rendering techniques.

II. Related Work

The problem of 3D reconstruction spans across a wide range of applications that utilize different strategies to obtain accurate outputs. Commonly, 3D reconstruction is solved through the use of multi-view geometry (MVG) in which several images are stitched together to form a mesh. The main downfall of this approach is that it requires large amounts of good quality input data to obtain a single output. To get an accurate output there needs to be enough images to observe around all occluded surfaces, such as under the chin and around hair that covers the face [2]. Additionally, MVG approaches are unable to interpret surfaces that are not perfectly diffuse because any reflections or refractions of light will change with the camera angle. Since these approaches require inputs that restrict the potential applications, current research has focused on neural network-based approaches to accurately infer 3D geometry from the inputs [2]. A discussion of some of these neural network-based approaches follows in this section.

A. Volumetric representations

Many studies focus on volumetric approaches to generate voxels to represent a 3D shape. These methods tend to adopt other information such as facial landmarks as the primary representation. The computational cost of these algorithms is very high, because they require lots of calculation with dense 3-dimensional structures. Additionally, usage of volumetric data requires large amounts of memory and storage space.

Jackson *et al.* used a CNN which they termed a Volumetric Regression Network (VRN) which extracts features from a 2D input and mapping them to a 3D

voxelization composed of roughly 7.4 million voxels [3]. The architecture of the VRN takes the form of two hourglass networks that work to establish a spatial correspondence between the input and output. The network calculates the 3D spatial predictions at the voxel level, which produced outputs that were robust to facial poses, expressions and occlusions. While this network has been shown to produce good results for 3D face reconstruction, it still lacks the crucial facial characteristics of texture and color. Additionally, the model creates a lot of useless information. The voxels behind the surface of the face do not serve much of a purpose, if any at all, because no information is able to be inferred about them, due to the opacity of the face. In our network, we work to ensure that all produced information is useful in order to minimize computational waste.

B. Surface-based representations

Surface-based representations such as meshes and point clouds can be used to get a detailed 3D shape. This approach is advantageous in memory efficiency because it does not have to store dense high-dimensional information. However, it is not easy to fit into deep learning architectures and several loss functions are often required to obtain accurate results.

The Pixel2Mesh paper uses a cascaded mesh deformation network as an end to end deep learning framework for 3D reconstruction [4]. The method involves beginning with an ellipsoid mesh and deforming it to approximate a 3D model of the input known as a graph-based convolution network (GCN). Next, the ellipsoid is deformed through a series of mesh deformation blocks that use graph-based convolution to extract features and update the mesh. After each deformation block there is a graph unpooling layer in which the number of vertices is increased to improve the resolution of the results. The graph unpooling layers allow for a coarse-to-fine approach, where the network is able to learn coarse shape representation, and fill in finer information later on in the pipeline. The loss function chosen for the GCN includes terms that account for chamfer distance, surface normal, laplacian regularization and edge length to ensure the shape of the ellipsoid converges to the desired geometry. Unfortunately, the Pixel2Mesh method does not provide a solution for texture and color of the meshes either [4].

III. PROPOSED METHOD

The overall goal is to map an image of a face to a realistic 3D mesh of the face. In order to achieve this mapping, the pipeline must be able to complete a few key tasks. It must be able to extract features and keypoints from an image of a face, map those keypoints and features into three dimensions, and synthesize a mesh to approximate the facial structure.

A. Preliminary Method

As a preliminary approach, we implemented methods of extracting face landmarks and segments from a 2D image, which can assist 3D face reconstruction, to explore effective methods.

1) Face Keypoint based Method

We implement a 3D face generation based on the key points which are extracted from a 2D image. First, the face area is detected by HOG features and a linear SVM classifier and face landmarks are detected using the face ROI. Face area and landmark detection is implemented with Dlib. After getting 68 landmarks from an image, shape fitting is implemented by using PCA shape coefficients and linear shape regression [5]. Also, facial components are used to calculate facial poses for reconstruction processing with Surrey Face Model. Surrey Face Model is used as a dummy shape of the model and it has about 3k vertices. Figure 1 shows a sample process of 3D face modeling based on the facial landmarks.

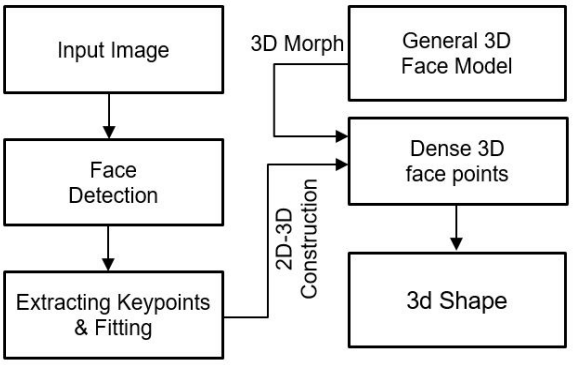


Fig. 1. Flow chart of 3D face modeling based on facial keypoints.

2) Face Segmentation

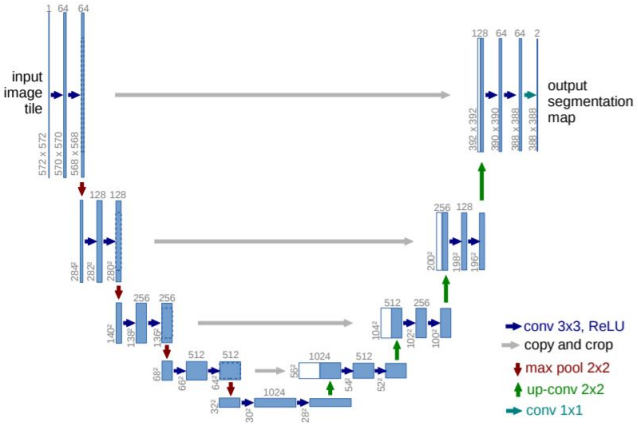


Fig. 2. U-net architecture for facial segmentation

Current methods of 3D reconstruction provide only an output for the geometry of the 2D input image. The goal of this project extends past obtaining accurate facial geometry and to forming a realistic face in terms of texture and color. The main obstacle that prevents a simple solution to this problem is that a face contains several areas such as the eyes, lips, and hair that are significantly different and interact with light differently, and also require different

levels of mesh vertex concentration. For example, the accurate representation of an eye and eyelid will require more mesh polygons per unit area than a forehead. To differentiate between different facial regions, a U-net architecture [6], shown in Figure 2, is used for segmentation. The specific facial features can be used to define the area of each facial element and to extract texture from the original 2D image. This segmentation technique appears promising.

B. Physics-based Face Reconstruction Method

In preparation for processing faces, a preliminary step is taken to provide a simpler example to ensure the feasibility of the method as well as to become familiar with PyTorch3D [7].

A Mesh deformation network was built and a physics-based loss function was implemented using a differentiable renderer from PyTorch3D. Also, the network uses facial segments which are extracted from U-Net to focus on facial area and to consider physical effects such as background area, illumination, and shadow. The framework of our proposed method is shown in Figure 3.

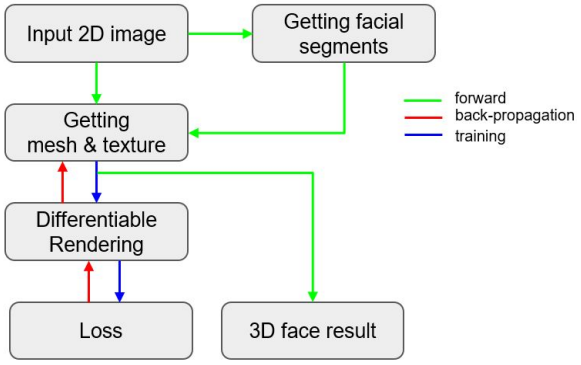


Fig.3. The framework of our proposed method for 3D face generation.

C. Mesh Deformation

A mesh deformation network is created using PyTorch. The goal of the network is to take in an image and output information about how to move each vertex to create the proper mesh structure. Wang *et al.* utilizes a cascaded mesh deformation network that provides an end to end deep learning framework [4]. The network deforms an ellipsoid, while learning where to put new vertices, and learning the relationship between faces. The new vertices are created in graph unpooling layers, which uses an edge based approach to ensure regular vertex distribution. Additionally, they utilize a coarse-to-fine method for mesh deformation [4]. However, the structure of the method is complicated and requires many different parameters for loss function and procedures for controlling the movement of vertices.

In order to simplify the architecture, our network will not be concerned with regressing the relationships between faces. The number of faces and their respective vertex connections will remain constant with reference to the initial mesh. Thus, the deformation network only needs to learn how to properly offset the positions of the vertices.

Four different loss functions are commonly used in mesh-based learning [4,7]. Chamfer loss, the formula for which is shown below in (1), takes into account the location error for every vertex in the mesh.

$$l_c = \sum_p min_q |p-q|_2^2 + \sum_q min_p |p-q|_2^2 \quad (1)$$

The normal loss, (2), measures the difference between the surface normal at each vertex in the produced mesh with the corresponding normal in the ground truth mesh.

$$l_n = \sum_p \sum_{q=argmin_q(|p-q|_2^2)} \left| < p-k, n_q > \right|_2^2 s.t. k \in N(p) \quad (2)$$

In (2), p is a vertex in the predicted mesh, q is the closest vertex to p in the ground truth mesh, k is the neighboring pixel of p, and n_q is the observed surface normal from the ground truth. This term allows the observed surface normals to converge to the desired geometry.

Next, two regularization terms are introduced to avoid the loss function converging to a local minimum. Laplacian regularization prevents vertices from moving too freely and encourages neighboring vertices to have similar movements.

$$l_{lap} = \sum_p \left| \delta_p' - \delta_p \right|_2^2 \quad (3)$$

where δ_p' and δ_p are the laplacian coordinate of a vertex.

Edge length regularization penalizes flying vertices that cause long edges.

$$l_{loc} = \sum_p \sum_{k \in N(p)} |p-k|_2^2 \quad (4)$$

where p is a vertex in the predicted mesh and k is the neighboring pixel of p.

The over loss function is a linear combination of each of the four losses, shown below.

$$l_{all} = l_c + \lambda_1 l_n + \lambda_2 l_{lap} + \lambda_3 l_{loc} \quad (5)$$

where λ_1, λ_2, and λ_3 are chosen hyperparameters.

This network architecture provides a computationally efficient method of mesh generation that will serve as a foundation for the experiments of this project. However, calculating all of this loss requires ground-truth knowledge of surface normals.

To explore Mesh deformation, we created two different networks, BabyMeshNet and MeshNet. BabyMeshNet was used for testing the effects of convolutional layers and the ability for a network to perform spatial regression of vertex coordinates. BabyMeshNet takes in a 128x128 RGB image, passis it through the convolutional layers, and outputs a 2562x3 tensor, which correspond to the xyz coordinates of the 2562 vertices in the mesh. Then, using PyTorch3D, the image is rendered, and the loss function is the mean squared error between the rendered image and the original input image. This loss function has the advantage that it does not require ground-truth knowledge of the mesh structure.

MeshNet was built with a structure similar to BabyMeshNet. Initially, MeshNet was trained with just mean squared error as well. However, no promising results were able to be achieved, so the loss functions had to be altered. The alteration of the loss function is further discussed in section V. Also, the biggest differences between MeshNet and BabyMeshNet are that the input to MeshNet consists of both the image and the segmentation map, and the output of MeshNet is a 2x2113x3 tensor. Similar to BabyMeshNet, one of these tensors consists of the xyz coordinates of the vertices, but the other tensor consists of the corresponding rgb color of that vertex.

D. Differentiable Rendering

A component of the loss function used to train MeshNet is a physics-based mean squared error. The output of MeshNet is used to deform the reference mesh to its new shape. This new shape is then rendered using the differentiable rendering pipeline of PyTorch3D [7]. Differentiable rendering is a relatively new research focus in computer vision that provides a method for use of rendering in deep learning applications. The required rendering pipeline in many approaches has many complicated and specific components that require a GPU. The PyTorch3D differentiable renderer offers a modular framework with compatibility with PyTorch and CUDA. This framework has been used to render the output of BabyMeshNet to produce an image similar to the input image, and is used for MeshNet as well.

IV. EXPERIMENTS

In this Section, the datasets used for our methods and results of preliminary test and intermediate results are explained.

A. Dataset Description

For the preliminary test, for face keypoint based method, we used Surrey Face Model (SFM) [8] which includes 3448 vertices and Basel Face Model (BFM) [9] which includes more than 50k vertices to implement keypoint based face modeling and segmentation. SFM has vertex information including 3 different coordinates within the facial area. And BFM has vertices including head and neck. The datasets used for training the face segmentation network are FASSEG [10] and HELEN [11].

B. Preliminary Result

1) Face Keypoint based Method

As the first experiments, we implemented a 3D face generation based on the key points extracted from a 2D image. As shown in Figure 4(c) and (d), 68 keypoints and estimated vertices were generated and its 3D face was created. However, these results were not achieved with deep learning, so no further pursuit of this technique was made.

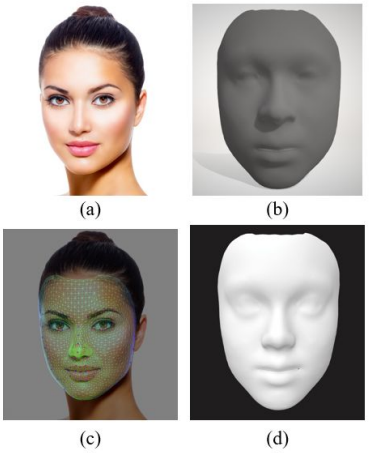
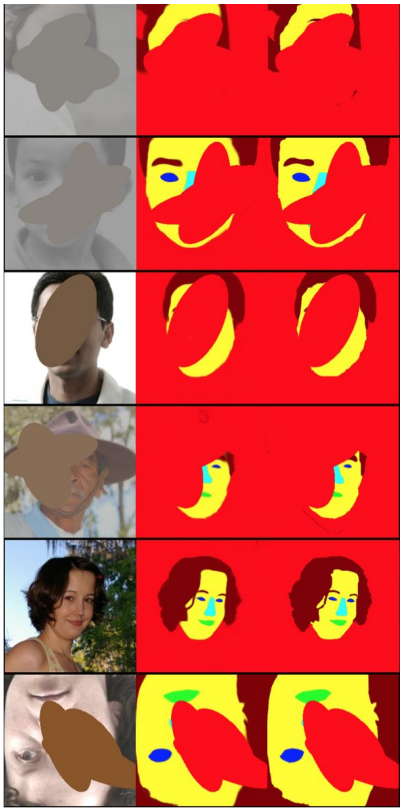


Fig. 4. Input and results of 3D generation based on keypoint: (a) input image; (b) Surrey face model; (c) result of detected landmarks and estimated vertices; (d) result of the reconstructed 3D face

2) Face Segmentation

Using a U-net based architecture [6], the segmentations in Figure 5 were achieved after approximately 24 hours of training. On the left are the input images, in the middle are the learned segmentations, and on the right are the ground truth segmentations. As can be seen, the segmentation is very robust and is effective even in cases of very heavy occlusion.

Fig. 5. Results of facial segmentation with U-net architecture: (a) input image; (b) ground truth; (c) results of facial segmentation



C. *Physics-based Face Reconstruction Result*

1) *Mesh Deformation based on BabyMeshNet*

To prototype mesh generation with BabyMeshNet, we used the teapot from the PyTorch3D tutorial. BabyMeshNet uses only 3 convolution layers with mean square error loss function. The image of the teapot is shown in Figure 6. BabyMeshNet takes in the image, and outputs a 7686x3 tensor, where 7686 is the number of vertices in the reference mesh and the three numbers in each row correspond to the x, y, and z components of the translation of each vertex.

BabyMeshNet uses 3 convolutional layers and is not set up with any U-net or ResNet enhancements, so the feature extraction is not highly robust at this time. Even so, enough features are able to be extracted to create a promising result, as discussed in the Section 3.

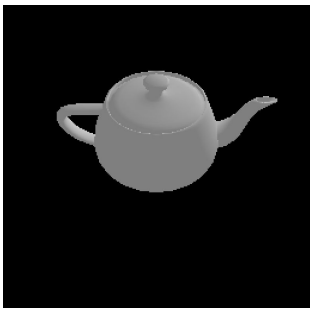


Fig. 6. Input image for testing with MeshNet.

2) *Differentiable Rendering*

A major problem that persisted for decades is that most renderers operate with a randomized non-differentiable method. In most cases, this does not pose a problem, but for deep learning, the non-differentiability meant that a renderer could not be used as a loss function. This severely handicapped deep learning in the area of 3D reconstruction. However, recently multiple packages capable of differentiable rendering have come under development, most notably Mitsuba2 [12] and PyTorch3D [7]. Mitsuba2 was explored for this project, but due to it still being in early development, it is not able to be utilized for our use case. PyTorch3D is slightly less realistic, but has an easy-to-use API, and integrates seamlessly with PyTorch.

Using the differentiable renderer from PyTorch3D [7], an image of the output mesh is created. The output of the renderer is an image that is the same size as the original. From this point, a loss function is calculated. At this point, the only loss function in use is mean squared error. There is no fancy feature-based loss or any advanced 3D mesh property or geometry-based loss.

As shown in Figure 7, there is an initial mesh, intermediate mesh, and the best mesh that was found.

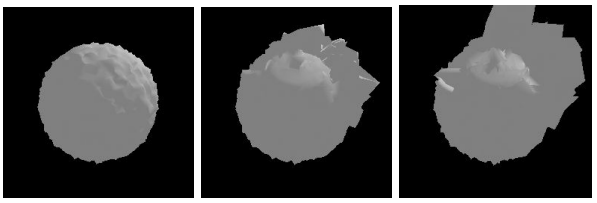


Fig. 7. Results of differentiable rendering with BabyMeshNet: (a) initial mesh; (b) intermediate result of mesh; (c) the best result of mesh

As can be seen in Figure 7, the deformation is able to be somewhat accurately learned merely from an image-based loss function. The top of the teapot can clearly be seen, and the spout and handle are beginning to emerge.

Given that this example is only run on one object, it is likely that the accuracy will increase tremendously when thousands of images are used for training.

2) *Mesh Deformation by MeshNet*

One the potential of BabyMeshNet was demonstrated, MeshNet was created. MeshNet consists of four convolutional layers and 2 fully connected layers, one for vertex position regression, and one for vertex color estimation. The reference mesh was a sphere. The vertices in this mesh were offset by the values output by MeshNet and the vertex colors were updated. The output image is then generated by using PyTorch3D's differentiable rendering pipeline. The structures of BabyMeshNet and MeshNet are shown in Appendix A and B respectively. MeshNet takes in the original image concatenated with the segmentation map (effectively 6 channels). At first, the network was trained with only mean squared error loss between the original image and the output image. However, this led to very blurry images that only minimally resembled a face. One such example is shown below in Figure 8.

Fig. 8. Original, terrible results

The vertices were flying all over the place, and the coloration was almost completely random. In order to mitigate this, we experimented with different loss functions. Unfortunately, the most powerful mesh-based loss function, Chamfer Loss, seemed like it would not be usable, because the ground truth mesh is not known. However, a hack was found. The reference mesh was changed to be a hemisphere instead of a sphere to allow for more informed vertex regression (the optimizer would no longer waste time trying to regress hidden vertices because they would now all be visible). Additionally, a chamfer loss was added between the final mesh and the reference hemisphere. Although the hemisphere is not the goal mesh, it is close enough to the

shape of a face that with a low weight, this loss could serve as a regularization term to prevent runaway vertices.

Even with this new loss and hemisphere reference mesh, the results were far from acceptable. At this point, the only loss was MSE with the original image, and Chamfer. To make the MSE loss more informative, the segmentation image was used to create a mask which got rid of the background in the original image and got rid of those same pixels in the rendered output image. Then, to improve training further, laplacian regularization loss, normal consistency loss, and edge length loss were added to make sure the surface was smooth and had relatively small triangles. Even with these loss functions, the network was still not learning very well. The final modification added was a VGG loss using a pre-trained VGG19 network. This was used to ensure feature consistency between the output image and the original image. The final loss function is below in (5).

$$l_{all} = 0.5l_{mse} + 0.6l_{VGG} + 15l_c + l_e + 0.01l_n + 0.1l_{lap} \quad (5)$$

Where l_{mse} is the mean squared error, l_{VGG} is the vgg loss, l_c is the Chamfer loss, l_e is the edge loss, l_n is the normal consistency loss, and l_{lap} is the laplacian regularization loss.

Even though all these loss functions were added, the result did not look realistic, although it did improve significantly. Some results are shown in Figure 9. as can be seen from the results, facial features are able to be found fairly accurately. These results are from 12 epochs of training. With more training, the accuracy would be further increased.

We used 3x64x64 images with batch size 1 for training and testing. The batch size limitation is due to a bug in PyTorch3d, which would cause the program to crash if a batch size larger than 1 was used. If larger batch sizes and higher size of the images are used, more accurate results will be obtained.

V. Conclusion

We implemented various methods for 3D face generation including face landmark extraction and face segmentation. Also, we presented mesh deformation and differentiable rendering with BabyMeshNet and MeshNet. As shown in experimental results, the 3D face model was created from 2D images. We used facial segments which were extracted by CNN. To get a 3D facial model, BabyMeshNet was created as an initial simple test. In order to improve the accuracy and represent more detailed 3D results with physical effects, we created a network, MeshNet, and the network was used with physics-based loss functions. As shown in the experimental results, 3D face was generated from a 2D image, even though the size of the input image is much small to represent details.

In order to improve our network for 3D generation, an image whose size is higher than 64x64 can be used for training and testing. Also, more complicated networks such as residual networks with larger parameters can be used to get more features. Additionally, graph convolutional networks could be explored, which may be much more efficient at learning relationships between vertices.

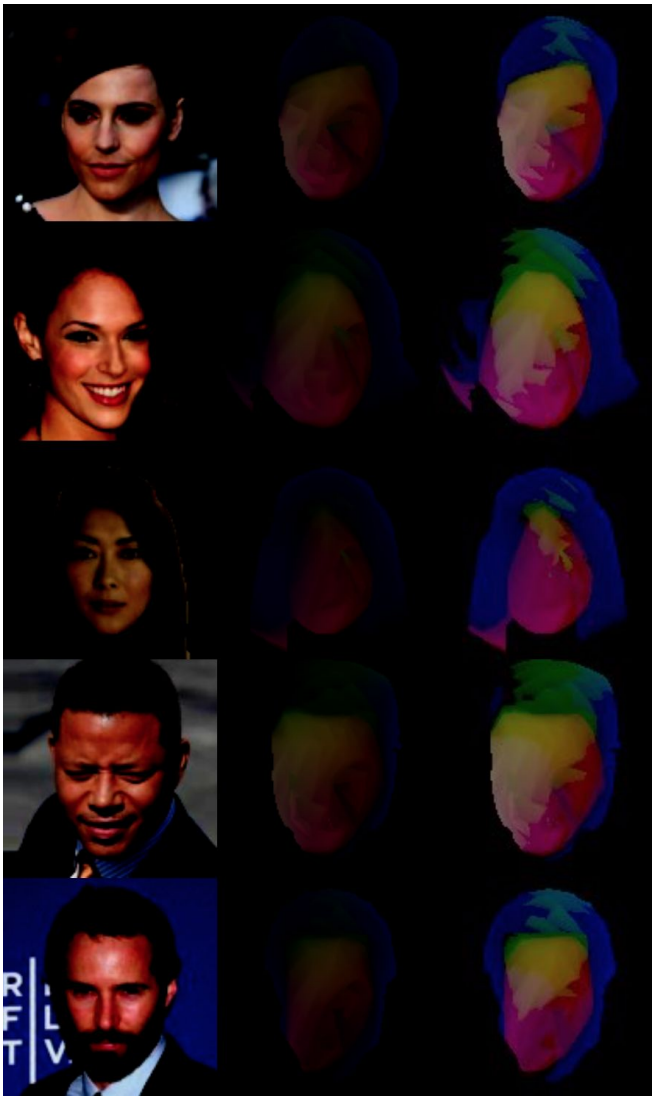


Fig. 9. Final results. Original Image, Rendered image, rendered image with higher contrast.

References

[1] V. Blanz and T. Vetter, "Face recognition based on fitting a 3D morphable model," *IEEE Trans. Pattern Anal. Mach. Intell.* 25(9), pp. 1063–1074, 2003.

[2] X. Han, H. Laga, and M. Bennamoun, "Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era," *IEEE Trans. Pattern Anal. Mach. Intell.* pp.1, 20 November 2019.

[3] A.S. Jackson, A. Bulat, V. Argyriou, and G. Tzimiropoulos, "Large pose 3D face reconstruction from a single image via direct volumetric CNN regression," in *IEEE International Conference on Computer Vision*, Honolulu, HI, USA, 21-26 July 2017, pp. 1031-1039.

[4] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.G. Jiang, "Pixel2Mesh: Generating 3d mesh models from single rgb images," In *European Conference on Computer Vision*, Munich, Germany, 8-18 September 2018, pp. 52-67.

[5] Y. Hu, D. Jiang, S. Yan, and L. Zhang, "Automatic 3D reconstruction for face recognition," In *IEEE International Conference on Automatic Face and Gesture Recognition*, Seoul, Korea, 17-19 May 2004, pp. 843-848.

[6] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," In *International Conference on Medical image computing and computer-assisted intervention*, Munich, Germany, 5-9 October 2015, pp. 234-241.

[7] PyTorch3D A library for deep learning with 3D data. Available online: https://pytorch3d.org/ (accessed on 10 April 2020).

[8] P. Huber, G. Hu, R. Tena, P. Mortazavian, P. Koppen, W.J. Christmas, M. Ratsch, and J. Kittler, "A Multiresolution 3D Morphable Face Model and Fitting Framework," In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, Rome, Italy, 27-29 February 2016.

[9] T. Gerig, A. Morel-Forster, C. Blumer, B. Egger, M. Lüthi, S. Schönborn, and T. Vetter, "Morphable Face Models - An Open Framework," In *IEEE International Conference on Automatic Face & Gesture Recognition*, Xian, China, 15-18 May 2018, pp. 75-82.

[10] S. Benini, K. Khan, R. Leonardi R, M. Mauro, and P. Migliorati, "FASSEG: A FAce semantic SEGmentation repository for face image analysis," *Data in brief*. 24, June 2019.

[11] V. Le, J. Brandt, Z. Lin, L. Bourdev, and T.S. Huang, "Interactive facial feature localization," In *European Conference on Computer Vision*, Berlin, Heidelberg, October 2012, pp. 679-692.

[12] Mitsuba 2 Physics based renderer. Available online: https://www.mitsuba-renderer.org/ (accessed on 10 April 2020).

Appendix A -- BabyMeshNet Model Summary

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1         [-1, 8, 254, 254]             224
          Conv2d-2        [-1, 12, 125, 125]             876
          Conv2d-3         [-1, 16, 60, 60]           1,744
          Conv2d-4         [-1, 32, 28, 28]           4,640
          Linear-5           [-1, 7686]          48,214,278
    SmallMeshNet-6         [-1, 2562, 3]                   0
================================================================
Total params: 48,221,762
Trainable params: 48,221,762
Non-trainable params: 0
```

Appendix B -- MeshNet Model Summary

```
----------------------------------------------------------------
      Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1        [-1, 16, 126, 126]             880
          Conv2d-2         [-1, 32, 61, 61]           4,640
          Conv2d-3         [-1, 64, 28, 28]          18,496
          Conv2d-4        [-1, 128, 12, 12]          73,856
          Linear-5              [-1, 6339]      29,216,451
          Linear-6              [-1, 6339]      29,216,451
        MeshNet-7  [[-1, 2113, 3], [-1, 2113, 3]]           0
================================================================
Total params: 58,530,774
Trainable params: 58,530,774
Non-trainable params: 0
----------------------------------------------------------------
```