ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

VLG Computer Vision and Learning Group

Leveraging Geometry-Aware GANs for fast Supervised 3D reconstruction of Human Face depth and Appearance from single-view images

Master's Thesis

Christos Antoniou
Department of Information Technology and Electrical Engineering (D-ITET)

Advisors: Dr. Sergey Prokudin
Supervisor: Prof. Dr. Siyu Tang, Prof. Dr. Marc Pollefeys

September 30, 2022

Abstract

Photo-realistic rendering is still a challenging task in the field of computer vision and computer graphics. Lately, there has been a wide community focus on 3D face avatar reconstruction, yielding impressive results for both video and image inputs, however extracting face shape representations from a single face image is non-trivial. Recent models, whilst successfully reconstruct facial details in 3D, fail to model hair geometry, whereas models that successfully model hair are mostly video-based and fail for single image inputs. Our model, addresses the problem of face depth and appearance reconstruction from a single-view face image. We present 2 methods, that leverage the generative power of large pretrained generative geometry aware models, to enable regression-based models, to infer face, hair geometry and appearance from single-view face images. The first method, is trained on high-quality synthetic face data and regresses depth values directly from a face image, by optimizing a depthmap reconstruction loss. Our method, achieves robust results in coarse and detailed 3D face reconstruction, by modelling facial details, as well as hair geometry. In comparison, to other methods that require require video inputs, our method directly reconstructs 3D faces during a single inference pass. Our depth-regression method bridges the trade-off between 3D reconstruction quality and inference speed, offering robust face geometry reconstruction on both synthetic and "in-the-wild" real data. The second method, regresses intermediate latent codes, for a geometry-aware GAN, directly from a single-view face image. Obtaining an accurate intermediate latent code, and by leveraging the GANs implicit pipeline, we achieve a volumetric representation of the entire face, along with its photo-realistic appearance. Our method, achieves good latent code reconstructions, in synthetic data, with performance deteriorating for real face data. Finally, our work, contributes "prosopo", a dataset of 1600 real human faces, with corresponding depthmaps and face meshes, for common baseline evaluation of 3D mesh reconstruction and depth prediction tasks from single-view face images.

Acknowledgements

I would like to thank Diego Arapovic and Yiming Zhao, for the constant emotional support and for allowing me to use their face in the experiments. Special thanks, to the AI Center, Companjon and Klaus Fuchs for providing the project inspiration and trusting me with carrying it through. I would like to, also thank Siyu Tang & the VLG group, as well as Marc Pollefeys for supervising the project. I dedicate this thesis to my beloved family, that is always there for me, supporting me through good and bad times. Finally, huge appreciation and gratitude to my advisor Sergey Prokudin, for all the constructive discussions and the continuous support throughout the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, there have been impressive results in computer vision tasks on 2D data. This spans from object detection, to semantic segmentation, classification, localization. The rise of GANs, has enabled to synthesize more realistic 2D data and extend computer vision and computer graphics applications, such as retargetting an actor/avatar, animating or reenacting. GANs [16] have contributed to a larger number of available 2D data and greater 2D vision applications, pushing the boundaries of 2D vision tasks to uncharted waters. Nevertheless, computer vision tasks involving 3D data are still challenging and a recent shift in community focus on tackling vision tasks on 3D world data, is aiming to bridge the performance gap. In contrast to 2D data, 3D data acquisition is more expensive and time-consuming. Obtaining 3D scans is a long process that involves hours of data capture with an expensive equipment. On top of that, post processing is required in most cases, to ensure the high-fidelity and homogeneity of data. In the internet world, where 2D data can be found in abundance, 3D data are scarce. This is not the only reason for the inferior performance in 3D vision tasks. Training on 3D data is computationally heavy, expensive and highly time-consuming, posing extra challenges for researchers or practitioners around the world. In this work, the focus is on 3D data, and specifically reconstructing a 3D face avatar from a single image, including hair geometry. In recent years, there have been impressive results on reconstructing human faces, obtaining the 3D avatar and even animating it. However, this impressive work sometimes cannot model hair geometry [13], or is mainly based on a video input [63], or on retraining the model with a single subject/actor [17] . Our focus, is to achieve a good 3D reconstruction only with a single input image of the human face. The aim is to achieve good results on both synthetic and real human face images (in-the wild images), without subject retraining and by deploying a simple depth prediction model for fast inference. Obtaining the geometry from a single image is a very challenging task, and belongs to the general field of inverse rendering. Previous methods have attempted to achieve high-quality 3D reconstruction results by leveraging 2D or 3D data. FLAME model [27], released in 2017, is the corner stone for many of the breakthroughs followed in face 3D geometry. It was trained on 3D data, specifically on 33,000 aligned face scans, providing the community with an generic expressive face/head model of learned shape space of identity variations, an articulated jaw and neck, and eyeballs that rotate. In 2019, DECA [13] a model using FLAME as its baseline achieved state-of-the-art results on 3D reconstruction of face geometry, just based on a single image. It regresses FLAME parameters, to influence the pose, shape, expression of a 3D face. It does not model hair (model interprets as shape deformations), however it models animatable face details that are specific to an individual but change with expression. Other work, based on the original 3D morphable models [12], the known ancestor of FLAME, combined with the powerful generative adversarial networks [16][10], allow to reconstruct robustly the shape geometry, as well as the texture from single images. This method [10], does not model hair geometry, however in similar fashion, the synthesizing capabilities of GANs are leveraged by more recent work called Neural Head Avatars [17]. This end-to-end model, models face, neck and hair geometry, as well as texture, with learning a specific subject. It requires a monocular RGB video input, to jointly optimize texture and appearance across the various video frames. An initial coarse shape estimate is

extracted using FLAME. A modified FLAME model is then leveraged by a geometry network (which is fed with modified FLAME parameters), which generates and corrects hair geometry. The generative geometry network, is fed with shape, expression and pose FLAME parameters, along with some mouth region specific parameters (modified FLAME), to generate 3D face shape with hair geometry. Recent work, related to GANs, enabled the synthesis of high-resolution face images, along with depthmap generation and face mesh reconstruction. This model, termed EG3D, is a geometry aware GAN, that introduces an expressive hybrid explicit-implicit network architecture, based on StyleGAN2 [25][24][23], that synthesizes high-quality images, along with its corresponding 3D geometry. EG3D can be leveraged for reconstructing any real image, with very high-quality results, by projecting/inverting the image to the EG3D latent space, using the method of PTI-inversion [41]. This method yields impressive results, however requires the camera parameters as input, as well as retraining of the PTI projector, leading to undesired long inference times. As part of this work, a model is trained on recovering intermediate EG3D latent codes from input RGB face images and hence directly leverage EG3Ds reconstruction and edditing abilities for any given single-view face image.

It is evident, that obtaining a high-quality 3D face shape with hair geometry, based on a single image is still a very challenging computer vision task. This paper aspires to address this task, by also achieving short inference times, to accommodate real-world applications. In the Figure below, a high-level summary of the model pipeline is depicted. By leveraging the synthesizing power of GANs, EG3D is used to produce synthetic images and

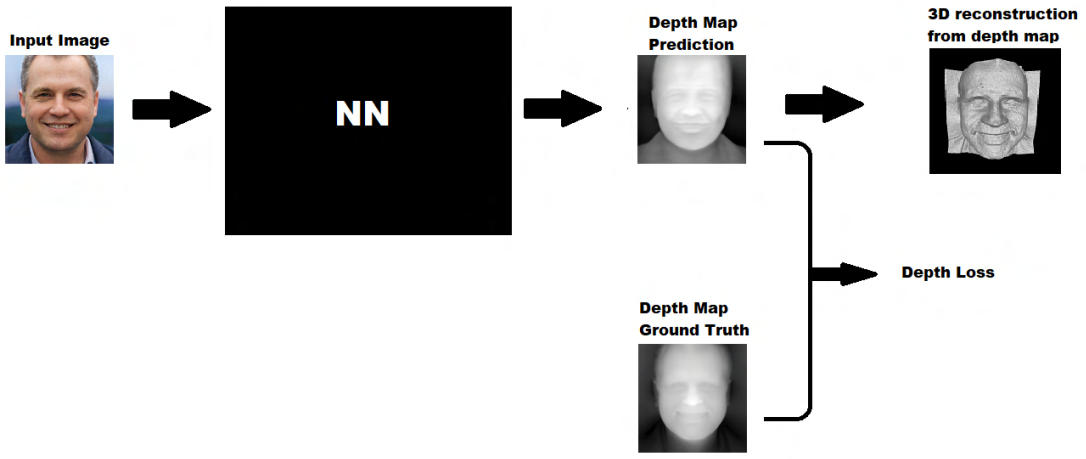


Figure 1.1: High-level framework pipeline. The framework is fed with a single RGB face image and its depthmap is predicted. The training of the framework involves optimizing the loss between the predicted depthmap and the ground truth map (GT). The final model, predicts the face depthmap during inference, which is later reconstructed as the 3D face avatar of the image face

their corresponding depth maps. This generates a vast amount of training data for our model. To train also, on real in-the-wild images, a PTI latent code inversion method is applied on EG3D latent representation, to generate depth maps for a real dataset such as FFHQ [24]. A real image is projected to compatible EG3D latent code and then is used to synthesize the initial real image, as part of the EG3D [10] domain. The low reconstruction error between the inverted and real image, allows us to leverage in-the-wild-images for model training. The inference stage, requires an single RGB image of a human face as input (a pose of a human looking straight to the camera), to produce a depth map. As a next step, a 3D reconstruction of a human face avatar, can be directly achieved using the depth map. During training, the predicted depth map, is compared to the ground truth depth map, and the pixel difference is minimized using a reconstruction loss.

Chapter 2

Related Work

The reconstruction of 3D faces is a long-lasting problem in the field of computer vision and computer graphics. For decades, several approaches attempted to model facial geometry and achieve 3D reconstruction from a single image. An overview of these methods is presented below.

3DMM

The first face 3D reconstruction method, presented by Blanz and Vetter [5] , gave birth to the field of 3D Morphable Models (3DMM). A 3D Morphable Face model is a statistical model, for face shape and appearance in a space with explicit one-to-one correspondences. In essence, it is a generative model for 3D face and appearance, that assists users in 2 unique ways: 1) A 3D face model is derived from a single image and 2) The shape and texture of the face can be modified in a natural way [5]. 3DMM work on the principle of transforming the shape and texture of some initial examples of 3D face models, into a vector space representation. Then by forming linear combinations of these faces, new faces and expressions can be modelled. Each individual face, is represented by a shape vector **S** and a texture vector **T** (Figure 2.1). In the original 3DMM model, shape vectors are formed by the 3D coordinates of its n vertices (x_i, y_i, z_i), whereas the texture vector is formed by the RGB color values of of each the n vertices (see Figure 2.2).

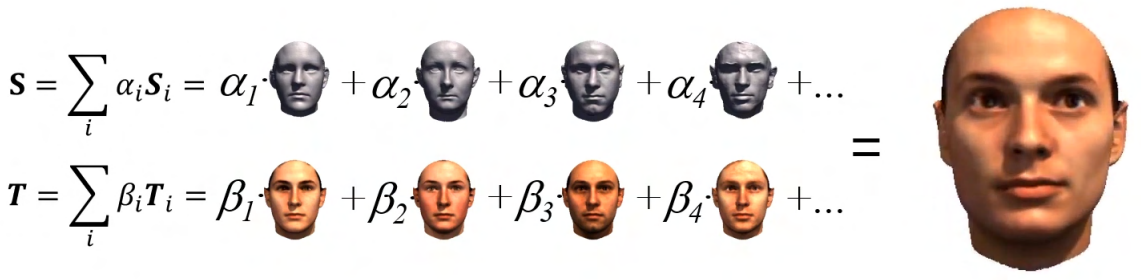


Figure 2.1: A Morphable 3D Face model, generates new faces (morphs) by a linear combination of other densely-corresponded faces. α and β are the parameters/coefficients of the shape and texture model respectively. Figure style adapted from [5]

The robustness of the model, is achieved by implementing some shape and texture constraints based on some statistical information derived from the set of example faces. To obtain a morphable 3D face model, we leverage large datasets of 3D face scans and establish dense correspondences between the example faces, as part of the so-called registration process. The registration process, ensures that the raw 3D face scans, are aligned uniformly, amongst all examples. The scans, are in essence reparametrized, such that there is a semantic and

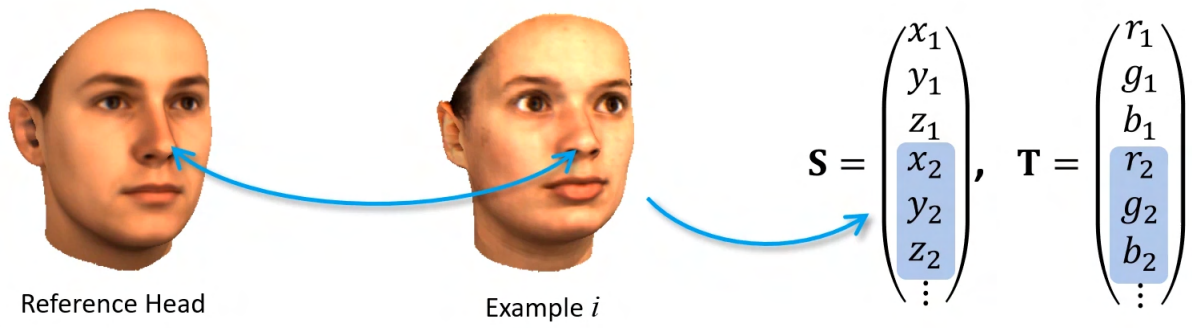


Figure 2.2: Each 3D face has a vector space representation **S**, **T** for both shape and texture. In order to form new faces, these vectors need to be aligned for all 3d examples of the dataset. Each vertex $v_i = (x_i, y_i, z_i)$ of reference scans and any new 3D example, needs to correspond exactly to the same face structure (e.g nose for v_2). The same applies for the color texture values (r_i, g_i, b_i). Figure adapted from [35].

parametric correspondence [35], meaning that each vertex v_i, corresponds to the same structure of the face, across all scans. For instance, the nose of the face, corresponds always, to the second position/entry in the data/shape vector (see Figure 2.2). Establishing dense correspondences is one of the most challenging tasks, for achieving a good reconstruction, as it is crucial in all morphing techniques [5]. By achieving good dense point-to-point correspondences, linear combinations of faces, are defined in a meaningful way, producing morphologically realistic faces (morphs) [12]. A morphable model is constructed by performing dimensionality reduction (usually PCA) on a set of 3D face examples, placed in dense correspondence. The morphable model can be used to create instances of new faces, by linear combination of other faces (from the training dataset), as well as reconstruct 3D faces from images. 3D reconstruction from a single image, is an ill-posed problem, however the morphable model, restricts the solution space, to morphological plausible faces. In the original work, 2 decades ago, 3D reconstruction was achieved by the so called analysis-by-synthesis, meaning reproducing a render image (with the applied 3D face model on top), that is as similar to the input image. The image difference between the reproduced image (requires rendering parameters such as the pose and illumination - combined with the 3D face model) and the input image, is then minimized by iterative non-linear optimization, for improved 3D reconstruction. The method of analysis-by-synthesis is also widely known as inverse rendering. To sum up, the ground breaking seminal work from Blanz and Vetter, is the first generic 3D face model - a statistical model for faces, that assists in 2D image-to-3D reconstruction, as well as providing a parametric face space to enable controller manipulation of the reconstructed face [12].

The implementation of this work, became publicly available in 2009 for research purposes as the **Basel Face Model (BFM)** [35]. The model was built from head scans of 200 young, mostly Caucasian adults, all in a roughly neutral expression [27] and 2 different linear models are developed separately for geometry and texture, due to independence assumption between face geometry and texture. BFM achieves good shape reconstruction, due to an improved registration algorithm, minimizing correspondence artifacts.

BFM, models facial geometry and expression, with a linear subspace, assuming a Gaussian prior distribution, resulting in a parametric face model with the mean , standard deviation and an orthonormal basis of principal components of shape and texture respectively [35]. By leveraging this statistical information and linear combinations of the principal components, naturally new faces are formed. The final BFM parameters, can disentangle pose lightning information from the identity parameters, improving face recognition and reconstruction.

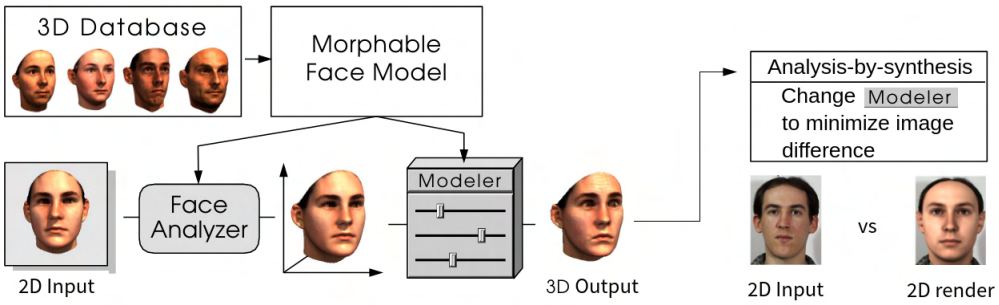


Figure 2.3: Visualization of 3DMM, a statistical model for single image 3D reconstruction of faces. The modeler allows also, the manipulation of the shape and expression of the reconstructed 3D face. 3D reconstruction is achieved by the analysis-by-synthesis method of matching the 3D output face model(rendered in 2D), with the 2D input image [5]. Figure style taken and adapted from [5].

Li et al. [27] introduced **FLAME**, an articulated expressive head model that provides nonlinear control over facial expressions by combining jaw articulation with linear expression blenshapes. Similar to BFM, FLAME is learned from thousands of accurately alligned 3D scans, however is more expressive and faces are learned with an articulated model and expressions. Specifically FLAME includes a low dimensional linear head shape face space, learned from roughly 4000 aligned scans. The registration process, ensures that the distance between the scan and the registered mesh is very low (¡1mm), to achieve better results. The basic FLAME rig, includes neck, articulated jaw and eyballs, in contrast to BFM and is gender based (female and male model). The face and head shape models are represented in a low dimensional space through the linear dimensionality technique of PCA. The first 49 principal components account for 99 % of the dataset variance, whereas 300 are the total number of principal components accounting for the entire variance in the training dataset. FLAME also learns poses and faces expressions, as it is trained on 4D scans of various poses and extreme expressions. It learns linear blend skinning (LBS) weights for the neck, jaw and eyeball rotations, joint positions for the neck and jaw, pose dependent corrective blenshapes, as well as expression blendshapes. Overall FLAME is a lightweight and expressive generic head model learned from over 33,000 of accurately aligned 3D scans [27].

Speech-driven Animation and Reenactment

FLAME allowed the blossoming of other areas such as speech driven facial animation [50][11], leading to **VOCA** (Voice Operated Character Animation) [11], the first speech driven realistic 3D facial animation model that is readily applicable to unseen subjects without retargeting. The VOCA model, is in the intersection of deep learning and 3DMM, whereby it receives as an input a speech signal and it regresses directly FLAME parameters. This way, for each input sequence, the face can be animated using the FLAME model, as the main generic head model. An overview of the VOCA method can be seen in the Figure below,whereby an encoder-decoder architecture is leveraged for regressing robustly FLAME parameters. Large Variability in facial shape, motion, and speaking style, motivates using a common learning space, leading to FLAME being part of VOCA's animation pipeline [11]. Overall VOCA generalizes well across various speech sources, languages, and 3D face templates and achieves a very realistic animation performance.

More recent work [40], on speech-driven animation, leverages the FLAME model in similar style to VOCA. **MeshTalk** [40] is a generic method created by Facebook Reality Labs, for generating full facial 3D animation from speech. It differs to VOCA, by using a deep learning approach based on a categorical latent space for facial animation, that disentangles audio-correlated and audio-uncorrelated information [40]. This results to

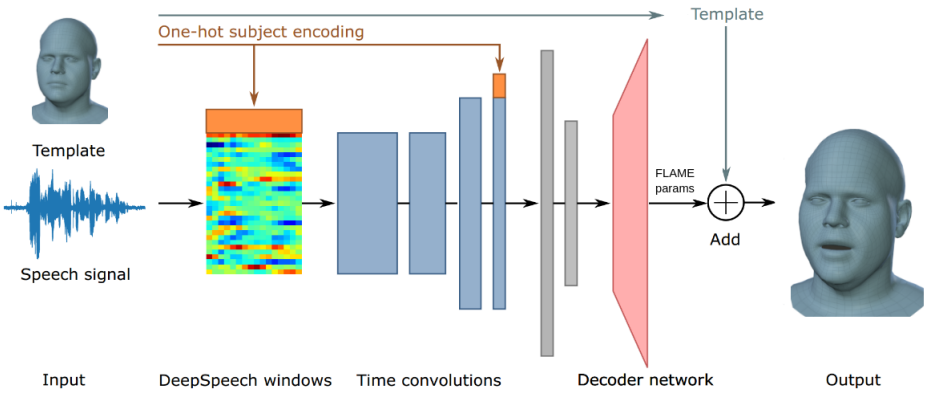


Figure 2.4: VOCA network architecture. The final output FLAME model (with extended neck), is shown on the right. The initial neutral face FLAME rig, is depicted on the left, before the predicted FLAME parameters (corresponding to a specific word) are applied. As observed, in comparison, to the BFM model, the FLAME model template, includes an elongated neck region. Figure from [11].

an overall more realistic animation, with highly accurate lip motion, better eye brow motion and synthesized eye blinks, replacing the more static details found in VOCA animations. Meshtalk, uses a categorical latent space representation (inferred by the encoder and the input speech signal), to directly generate an animated mesh, in contrast to another method similar to VOCA, called **"Neural Voice Puppetry"** [50]. This method, leverages an audio to expression network, that directly predicts coefficients (expression vector), that are fed to an intermediate 3D model and drive a person-specific expression blenshape basis. An extension to VOCA, is that the intermediated 3D model, is directly rendered via a light-weight neural rendering network, allowing not only for reenactment, i.e., transferring the face motions from one person (source) to another (target), but also speech-driver photorealistic video synthesis. It is evident that the area of reenactment has benefited from the creation of 3DMM, however, similarly to the 3D reconstruction area, deep learning and specifically generative models, have come into play. This enabled another reanactment method [55], leveraging a video-to-video synthesis model, specifically trained on faces, that uses GANs to map an input video (driver video) to an output photorealistic video of the source video content (e.g an photorealistic video of the American president, with the same facial expressions or mouth movement with the driver video). The framework from NVIDIA, termed **fs-vid2vid** [55] does not leverage 3DMM, however it demonstrates, that 2D facial reenactment has reached a very photorealistic level, achieving realistic facial expressions, paving the way from moving past 2D neural head avatars [55][59] to 3D avatars, with the aid of generative models.

Detail/Animatable 3D reconstruction

Lately, 3D Morphable Models have been rediscovered in the context of deep learning, leading to 3D reconstruction of 3D faces with much more details. The methods described above are based on 3DMMs that achieve monocular 3D reconstruction by estimating coefficients of statistical models (e.g FLAME, BFM) via analysis-by-synthesis techniques. These methods, result to modeling 3D faces with low frequency face details (such as nose, ears, mouth), mainly the head and face structure, achieving a so called coarse 3D reconstruction. Smooth reconstructions, do not lead to realistic results and the need for more medium frequency information capture, is necessary to improve reconstruction quality. A novel method, named DECA regressed 3D face shape with animatable details that are customized to an individual, but change with expression [13]. It reconstructs a detailed shape by avoiding artifacts, by modelling explicitly dynamic parts of the face, such as wrinkles. DECA

learns to regress 3D animatable faces, directly from in-the-wild training images. DECA is an end-to-end deep learning method that leverages 3DMM and is based on the assumption that static face features such as the overall shape of the face, stay the same for each person (identity assumption). On the other hand, DECA follows the principle that dynamic features, such as wrinkles, change with expression, to be modelled as well, for better reconstruction. An overview of the DECA pipeline is show in Figure 2.5.

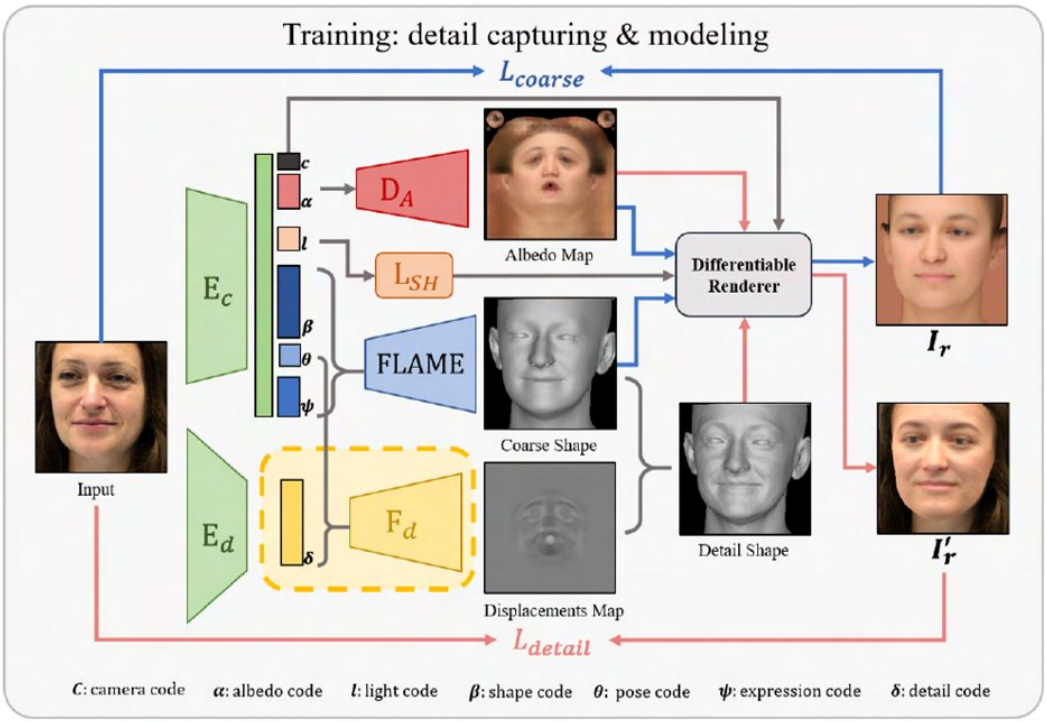


Figure 2.5: DECA framework pipeline. During training, DECA optimizes 2 main loses, the coarse reconstruction loss (blue arrows - L_{coarse}) and the detail reconstruction loss (red arrows - L_{detail}). Both losses are optimized by comparing the input image to the coarse rendered image I_r and the detailed rendered image $I_r{}'$ respectively. The analysis-by-synthesis procedure is a community standard, however the key novelty of DECA is the network - F_d (yellow) that predicts the displacement map. DECA enforces a novel detail consistency loss, that disentangles expression(ψ) from detail code (δ) In general E_c, E_d are the image encoder networks (Resnet50) and D_A, L_SH are the model for albedo map extraction and landmark detection, respectively. Figure from [13].

As a high level overview, initially an input 2D image is passed through an image encoder (Resnet50). The encoded code of the image, is then regressed to various parameters, specifically to FLAME model parameters (β, θ, ψ), landmark detection parameters (l), displacement map parameters (δ) and albedo-map parameters (α) (see Figure 2.5 for parameter explanation). These parameters are optimized with respect to a coarse reconstruction loss - L_{coarse} and a detail consistency reconstruction loss - L_{detail}. In essence, the model initially reconstructs a coarse shape of the face. This coarse representation is optimized by minimizing the distance between the input 2D image and the coarse shape photorealistic rendered image I_r. At the same time, a detail shape reconstruction is produced by implementing the predicted displacement map on top of the coarse shape. This detail shape is rendered into $I_{r'}$. Then by minimizing the distance between this render image and the 2D image, the detail shape is optimized, producing realistic 3D reconstruction of the input face. DECA is the first method that successfully models animatable details such as wrinkles, however it cannot model glasses and hair, as hair and glasses are interpreted as shape deformation of the FLAME model rig. The model lack robustness in occlusion examples, however there is a detailed reconstruction, with quality images. An example of the

reconstruction performance with animatable details is shown in Figure 2.6. Finally, the method can achieve reposing or face reenactment, such as expression transfer from a 2D image, for a source expression to a target identity (see Figure 2.7. To sum up DECA is an end-to-end deep learning method trained from 2M in-the-wild face images without 2D-to-3D supervision that achieves a fine detailed shape reconstruction, by disentangling identity and expression-dependent wrinkle details.

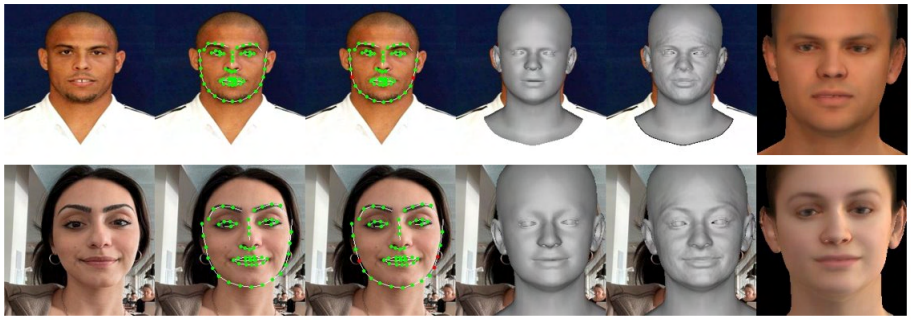


Figure 2.6: Visualization of DECA reconstruction from a 2D image (column 1), to a coarse (column 4) and detail shape respectively (column 5). The face landmark detection is depicted in columns 2 & 3, along with the rendered image in column 6.

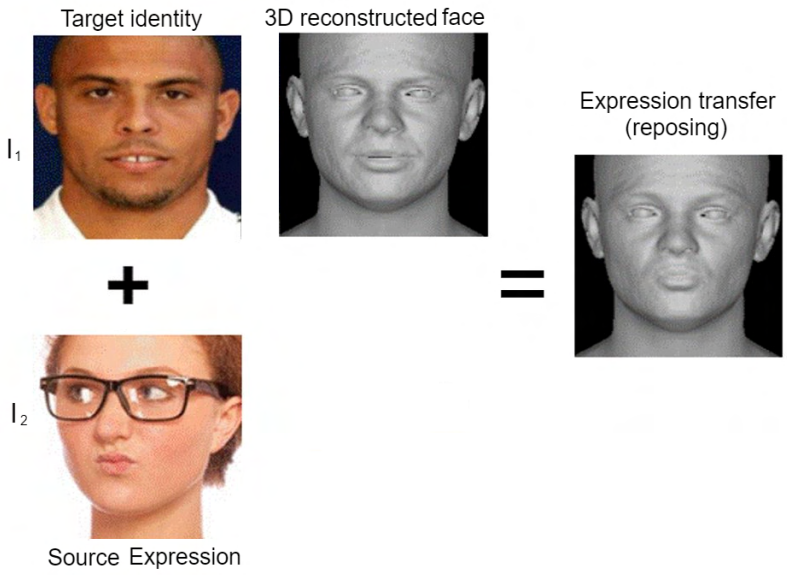


Figure 2.7: DECA can transfer a source expression to a target identity, by leveraging the disentangled expression (wrinkle-dependent) and detail (person-specific) code. Reposing through DECA is achieved by replacing the expression code of the target image (target identity) - ψ_1, with the source's (source expression) expression code ψ_2.

Generative Models

Deep learning has enable detailed 3D reconstruction from a single 2D image, resulting to textured 3D avatars, with great photo-realism. Newer methods, have successfully reached this level of photorealism, through the aid

of recent deep learning advancements, known as GANs (Generative Adversarial Models). GANs are powerful synthesizing tools, that have assisted the field of computer vision in the last decade.

GANs are algorithmic architectures, that consist of 2 neural networks, the discriminator and the generator, that play an adversarial game on a certain training set. For instance, in a face training set, the generator aims to produce as realistic human faces as possible, so that the discriminator does not detect them as fake synthetic faces. GANs are trained by respectively fine-tuning the discriminator and generator networks, by optimizing a minmax GAN loss, to obtain powerfull and realistic synthetic face images. The content of synthetic samples generated, relies on the set of training examples. **StyleGAN** [24] [25][23] is a novel GAN architecture borrowing style transfer literature, to synthesize large high-quality images of faces. StyleGAN is an extension of the progressive growing GAN [22], a method for incremental expansion of both discriminator and generator models, to account from small to large images during training. In each step, the generator and discriminator expand in size, and are fit until stable. Once stable, the training image's size (width, height) double, following the models expansion. Hence, StyleGAN steadily synthesizes large fine detailed high-quality images [24]. A main difference with the original [22] progressive GAN architecture, is that the neighboring layers, are replaced by bilinear upsampling layers as the main method for expanding the network. An additional contribution of this method, is the introduction of a mapping network. Instead of feeding a latent vector, randomly sampled from a latent space, the latent code z, is transformed to a "style latent vector". Each style block in the synthesis network g, applies this "style latent vector" after applying an affine transform (AdaIN - adaptive instance normalization). Then Gaussian noise is added to this transformed style vector, before its normalized, which allows StyleGAN to control the style-level variation of specific face details (e.g hair) . This summarizes, every block in the synthesis network g (see Figure 2.8

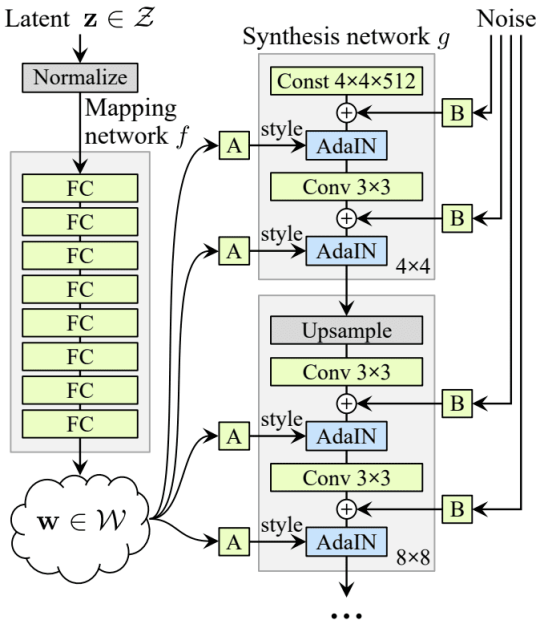


Figure 2.8: StyleGAN novel architecture. A latent single vector - z is sampled from the general GAN latent space. The latent code z is then mapped to a style vector w - an intermediate latent space, via the mapping network. This is different from a traditional GAN architecture [22], that receives just a single latent vector as generator input. In the diagram "A", stands for affine transformation and "B" refers to a per-channel scaling of the noise input. The style vector, is incorporated through adaptive instance normalization (AdaIN) at each convolution layer, after an initial affine transformation is applied. Figure from [24].

Style latent vector control the style of the generated face. Specifically, they allow an "unsupervised seperation" of high-level attributes"- such as pose or identity - and the "stochastic variation" in the generated images,

such as freckles or hair. As a result, style latent vectors fed to the lower layer blocks control the large-scale styles, and those fed to the higher layer blocks control the fine-detail styles [24]. Finally, StyleGAN achieved a better latent disentanglement of facial features, hence moving across a latent direction (linear hyperspace in the latent space), influences specific facial attributes (binary) (e.g hair or not hair). This has enabled better also, better interpolation.

In general, StyleGAN has a wide variety of applications, such as facial editing, face generation and face disentanglement. Other methods such as **StyleRig** have built on top of StyleGAN, enabling face expression, pose, illumination manipulation via a 3DMM [48]. It provides a a rig-like control over semantic face parameters that are interpretable in 3D (e.g face expression) via 3DMM, combined with photorealism offered by StyleGAN. A new rigging network, named "RigNet" is trained between the 3DMM's semantic parameters and StyleGAN's input. By learning a mapping between the parametric space of 3DMMs and StyleGAN, a rig-like control of facial attributes is achieved by preserving StyleGAN synthesizing capabilities and photorealism. Simply stated, it translates semantic edits on 3D face meshes to StyleGAN's input space (new w style latent vector) [48]. As shown in Figure 2.9, rigging the 3DMM's parameters, the synthesis procedure is controlled, outputting a portait StyleGAN-style face image with the desired facial expression.

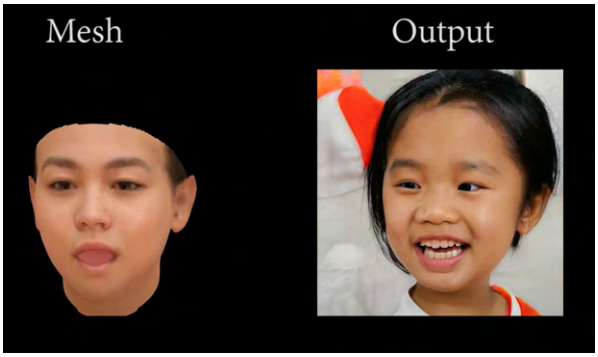


Figure 2.9: StyleRig high-level principle. Rigging the semantic parameters of the 3DMM, allows controlling the semantic parameters (e.g expression) of the synthesized human face. Figure from official presentation video of [48].

StyleRig still fails to exploit the full expressivity of the parametric face model, as well as control the background or hair style, as it is not modelled by a 3DMM. Limitations like these, or the lack of fine-scale detail reconstruction control, are solved by more recent advancements in the field [9].

Most generators synthesize images, however a recent generator model named **EG3D** [9], synthesizes high-quality realistic human faces, along with the corresponding depth maps and 3D shape. EG3D is the first geometry-aware generator, that can synthesize 2D high quality face images, along with their 3D shape reconstruction. A key novelty, is that it can reconstruct all the facial details, along with hair, glasses and any other facial attribute of the synthesized image. To achieve a 3D reconstruction from an custom 2D input image, a GAN inversion method is applied on the input image [41]. In essence, the input image, is transformed to the generators latent style vector, before synthesizing the respective 3D shape, depthmap and 2D input image (reproducing). EG3D, is a 3D GAN [56] that produces photorealistic images that are both multi-view consistent and geometry-aware. It enables supervised learning of high-fidelity 3D represenations without explicity 3D or multi-view supervision. EG3D combines an "efficient explicit–implicit neural representation with an expressive pose-aware convolutional generator and a dual discriminator" [41]. An in-depth overview of EG3D, is presented later in the method section (**??**).EG3D achieves good 3D reconstruction of the face shape, by emphasizing on good geometric awareness, resulting to the convexity of eyes (go inside, rather than outside) (see Figure 2.10. Even though, there is no 3D or multi-view supervision, EG3D needs the camera pose as input (pose conditioning), as it assists the model to decouple/disentangle appearance/facial expression from the pose/view.

Recent work, from Gecer et.al [14], achieved 3D reconstruction from a single image, by leveraging also

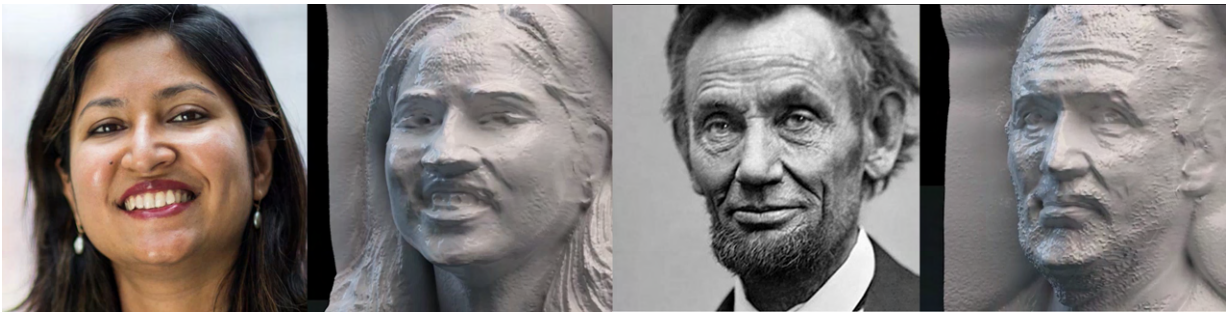


Figure 2.10: EG3D shape reconstruction of human face. This is a fine-detailed reconstruction, that can model hair, wrinkles and any facial details. Figure taken from [9].

generative models. **GANfit** uses a combination of GANs and DCNNs (Deep Convolutional Neural Networks), to reconstruct the facial texture and shape from single images. Specifically, a generator of facial texture in UV space is trained, to assist in reconstructing correctly the facial texture. Through an end-to-end differentiable framework, 3DMM fitting is carried out by optimizing the latent parameters, that best reconstruct the input image,by enforcing the supervision of pretrained deep identity features. Training is achieved by optimizing the distance between the 2D image and the rendered image (colored mesh aligned with image). A cost function that is based on face features (from face recongition) and features from the landmark detection network, is optimized by updating the latent parameters of 3DMM, camera pose, lighting and the texture network. GANFit also demonstrates how differentiable renderers (trained end-to-end), allow the exploration of facial identity features and all the rendering related parameters, (3DMM shape,texture, camera pose, lighting).

Another method, termed **AvatarMe** [26], reconstructs photorealistic 3D faces with high level of detail, by training on a large dataset of "in-the-wild" images (high variance in shape, lighting, reflectance). The framework, uses a traditional 3DMM based reconstruction pipeline - called LSFM [7][6] (similar to FLAME), to initially reconstruct the base geometry and GANFit to extract the texture of the 3D face. The method, achieves a photorealistic result, with fine-details, by additionally extracting not only diffuse albedo (DA), but also , specular normals (SN), diffuse normals (DN) and specular albedo (SA), to inferred reflectance of the face. The diffuse albedo, is obtained by,parsing the enhanced texture(via a super-resolution network) [62][58][26] through a de-lighting network [15]. Then the rest (SN, DN, SA) are inferred by seperate networks, by leveraging the diffuse albedo and the 3DMM shape normals. Overall, photorealistic rendering of the human face skin, with high frequency details is achieved. The results are very promising, nevertheless, the resulting 3D face shape/geometry does not model hair or similar face details.

Other approaches, such as **Neural Head Avatars** (NHA) [17], reconstruct 3D shape and appearance leveraging also Neural Networks (MPLs), by mainly focusing on monocular RGB videos. Even though, the method cannot reconstruct 3D shape directly from a single image, the reconstruction results and detailed pipeline are quite interesting. A hybrid representation is used, that leverages a morphable model for the coarse shape and expressions of the face, as well as 2 feed-forward networks, that predict geometry and appearance. The first feed forward network predicts vertex offsets of the underlying mesh, whereas the second network predicts a "view- and expression-dependent" texture. Following closely the standard graphics pipeline, NHA, uses an MLP geometry network G, that outputs a triangular mesh with V and F, corresponding to the vertices and faces of the mesh. The geometry refinement network introduced a pose-dependent offset function, that enables hair and facial detail modelling. The initial FLAME template, is adapted to include more vertices, to accommodate for the more facial details and hair. Once, the detailed 3D mesh is obtained, the texture MLP network T, predicts an RGB color value for each surface point of the output mesh (surface defined by V and F). The T network inputs the 3D coordinates of a mesh point on the adapted FLAME template mesh, the FLAME parameters of expression and pose, as well as a "local patch of the rendered normals", to output a predicted color value for this specific mesh point. This results to a photorealistic synthesis of the face texture, including pose

and expression-dependent effects. An overview of the method can be shown in Figure 2.11.

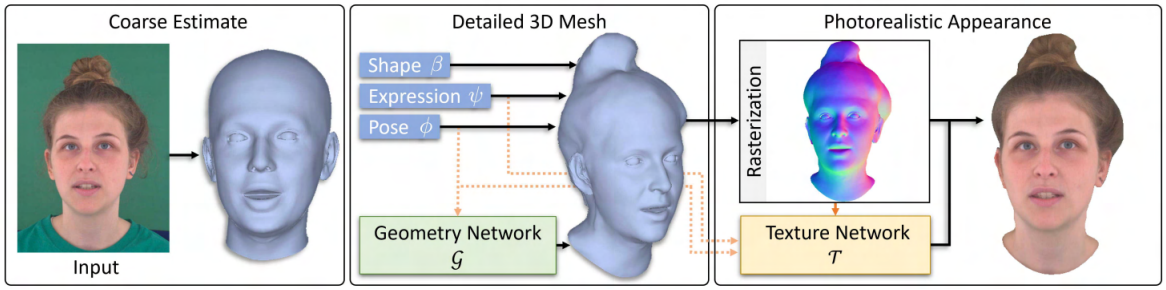


Figure 2.11: NHA high-level overview. Given an input RGB frame, the FLAME coarse 3D shape is estimated. All the FLAME model parameters are estimated, such as the expression (ψ), pose (ϕ) and shape (β). The pose flame parameters are fed into the geometry refinement network G, that outputs the offset vertices, that result to a full head geometry mesh. A triangular mesh with hair and extra facial details, termed as detailed 3D mesh (adapted FLAME). The pose and expression parameters (ψ and θ) are then fed to the texture network T, along with the rendered output triangular mesh (rasterization method) to produce the final textured render. Figure taken from [17].

Optimization of the 2 networks networks and FLAME parameters is carried out in analysis-by-synthesis method with geometry-related and appearance (color) related energy terms. The 2 energy terms are jointly optimized, however this energy term division, motivates the disentanglement of shape and color detail. The geometry objective, optimizes energy terms related, to a) a landmark 2D detection l1-loss b) a normal image Laplacian distance c) a semantic distance for facial skin, neck, eyes, nose d) regularization of the G network and FLAME parameters. On the other hand the appearance objective, consists of energy terms related to a) Dense per pixel distance from input image and render b) Perceptual distance (compare directly features of images extracted from face detector).

Implicit 3D reconstruction

The last few years, implicit functions have been widely used, in all areas of 3D computer vision [43] [32] [8]. A method called NeRF (Neural Radiance Fields) [32], revolutionized 3D reconstruction, through an "instance-specific implicit representation" of an object. NeRFs use a volumetric representation of complex scenes, to achieve a 3D reconstruction of the entire scene, based on a few images of different scene views. NeRFs optimize a continuous volumetric scene function F_θ, and are able to represent detailed scene geometry (e.g face shape) even with complex occlusions. The model architecture inputs a single 3D point in the scene(3D location - (x, y, z)) from a specific viewing position (2D viewing direction - (θ, ϕ)), and it outputs the predicted color of that pixel location (RGB color (r, g, b)) and its volume density σ (probability of an object being present at this location). The model architecture consists of an MLP of 11 fully connected layers, that is leveraged to overfit the neural network to the particular scene. Hence, by going over all 3D points, from all viewing directions, a full 3D representation of the scene is achieved. So for a set of sparse input images (see Figure 2.12), for a specific pixel location, and the particular viewing angle of the camera, a viewing ray is shot and random points across this ray - (x, y, z) are sampled. So for each sampled point, the MLP (F_θ) predicts its color and density. After, predicting the color for all these sampled points across a single ray, classic volume rendering techniques are leveraged [21], to calculate a weighted average of the pixel color. Hence, the color of a particular ray is computed, by integrating the density and color for each point along this ray. Once the predicted color for this 3D location is obtained, it is is compared to the ground truth pixel color (g.t) of the input image. As a result, the neural network is optimized using a reconstruction loss between the render color and the input image color. All the information for a single scene is encoded in the weights of the MLP. Even though, the entire scene is

encoded in a relatively small number of weights, for novel-view synthesis, to render a single pixel, the MLP is queried, across all ray points to get a single pixel in the image space (8 forward passes for single pixel rendering) . This is repeated for all pixels in the image, leading to a time-intensive operation for rendering a single novel view.

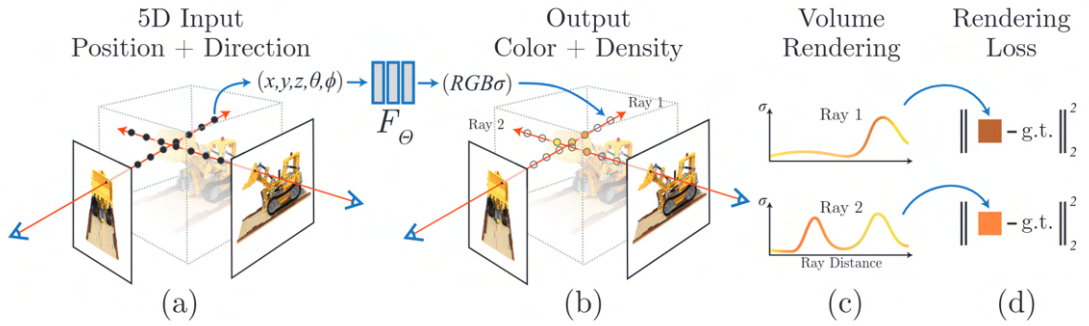


Figure 2.12: An overview of neural radiance field scene representation and rendering. NeRF pipeline involves sampling 5D coordinates (3D location and 2D viewing direction) along a traced ray (see a). Each location (x, y, z) across a ray, is fed in an MLP (F_θ) to obtain the color (RGB) and volume density (σ) (see b). By implementing the ray tracing method as a volume rendering technique, to composite the predicted color and volume densities across the ray points, into a single pixel image value (see c). The volume rendering function is differentiable, allowing optimization of the scene representation by minimizing the rendering loss, the pixel color difference between synthesized and ground truth images (see d). Figure is taken from [32].

More details about implicit representations and NeRFs can be found in the method section, at the in-depth EG3D explanation .

Recent methods such as ImAvatar [63] and LOLNeRF [39], have shown promising results in the field of 3D reconstruction, using implicit representations.

ImAvatar [63] focuses on learning implicit morphable head avatars from RGB video. This 3D morphing-based implicit head avatar model reconstructs detailed geometry and appearance with the ability to extrapolate to unseen expression and poses. It models both hair and facial details, via a neural volumetric representation, and offers the facial attribute controlability of 3DMMs (fine-grained control e.g expression), by incorporating morphable models. FLAME is used as the main morphable model template and enables the reconstruction of a 3D avatar, by modeling the canonical shape, texture and deformation fields of it. This solves, the problem of neural implicit models, that even though can model hair, mouth interior and more facial details, they lack the fine-grained expression control and facial geometry deformation-ability.During mesh morphing, the FLAME model, deforms the canonical mesh, by adding the expression and pose-correctives, before LBS is applied, to obtain the final mesh [27]. ImAvatar, represents the blendshapes and skinning weights as continuous fields and formulates the deformation in a similar way to FLAME, via implicit morphing (see Figure 2.13. Given a target pose and expression parameters, we can calculate the deformed location for each canonical point of the mesh. This is the principle behind implicit morphing. We first perform ray marching in the deformed space for each point along the view ray. Then for each deformed point, we need to locate its correspondence in the canonical space, where the geometry field is defined. Therefore, we leverage correspondence search to find the canonical location via an iterative manner, through our implicit morphing formulation. Next, the occupancy value of the sampled points is determined, by querying the geometry network and locating the nearest surface intersection for each pixel.

Once the canonical surface point is obtained, the texture network is queried, to obtain the respective RGB value for the pixel. This is done by first deriving the analytical gradient for the canonical point, to avoid back-propagation through the iterative process. As part of the novel gradient method, the gradient computation is carried out with respect to 2 constraints that ensure that a) the canonical point is on the surface and b)the pro-

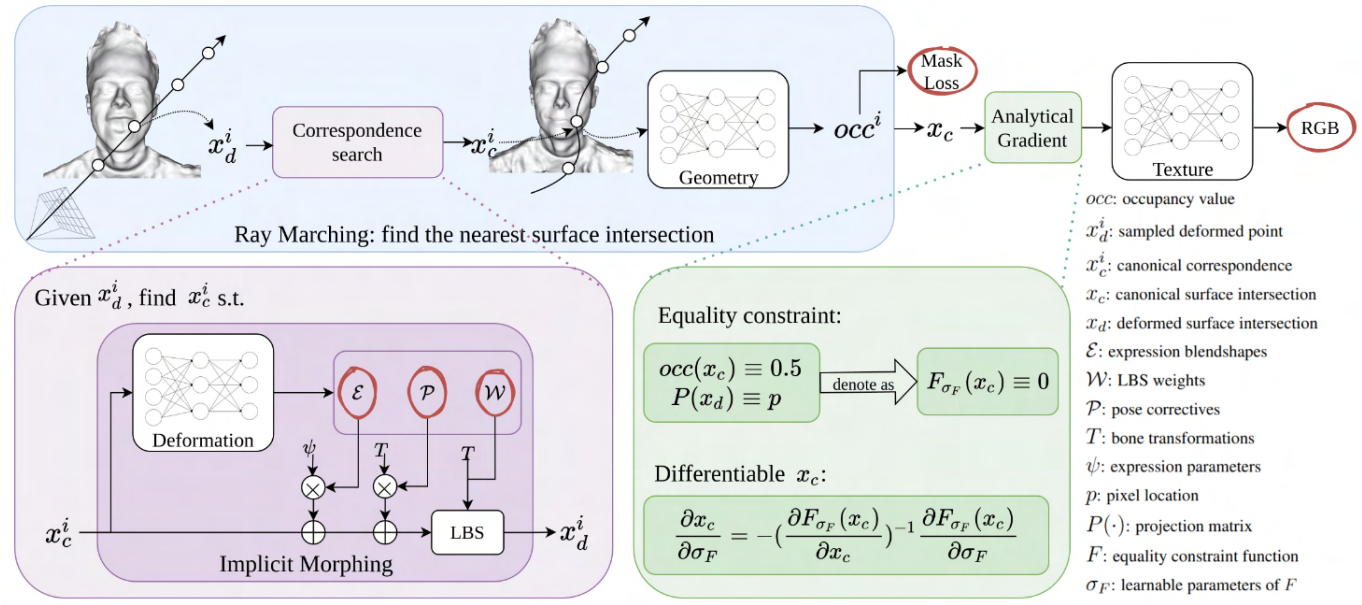


Figure 2.13: An overview of the ImAvatar method is depicted. Given a pixel location, the method performs ray marching in the deformed space, to find the nearest surface intersection between the deformed mesh and the canonical mesh. This is achieved through the novel implicit morphing, that updates the canonical blenshape and skinning-weight fields, to recover the deformed corresponding point. After finding the nearest canonical surface intersection x_c, we apply some equality constraints on the gradient calculation, that speed up the gradient computation for the geometry and deformation fields. Hence, after the analytical gradient calculation, the canonical texture network is queried, to obtain the final RGB value. Figure is directly taken from [63]

jected deformed point, projects to the pixel location. During training, we optimize an image reconstruction loss and a mask loss, by indirectly supervising the predicted blendshapes and skinning-weights [13]. Hence, ImAvatar is a differentiable rendering approach, that allows end-to-end learning from RGB sequences of frames. The approach uses volumetric representations [32] and requires a set sparse of examples from various viewpoints. As a result, during training an avatar, a video of the target face from a variety of viewpoints is required. In summary, IMAvatar is parameterized by the canonical texture, geometry and deformation fields.

Neural Radiance Fields have inspired, more recent work [39], that does not require multi-view data to generate realistic face images along with their 3D structure. **LOLNeRF** is a generative 3D model, based on neural radiance fields, that learns 3D shape just ”from one look” of a face. It implicitly learns a shape and appearance space, through reconstructing a large pool of single-view face images (or general objects), aligned to an approximate canonical pose. This is achieved, by leveraging a single neural network conditioned on this shared latent space [39]. Specifically, it learns a table of latent codes, where each latent code corresponds to one single-view image. At the same time, it optimizes the parameters of a foreground and background NeRF, to be able to decompose background and foreground details during generation. This method, requires a dataset of face images, with approximately aligned cameras, which is obtained through a ”least-squares fit between 2D landmarker outputs and the face canonical 3D keypoints” [39]. This helps, also extract the camera parameters that are necessary for the volumetric rendering part of the model pipeline. Similarly to ImAvatar, this method is also trained by optimizing an image reconstruction loss and mask loss, with the addition of a hard loss. nevertheless, the volumetric output depends on a per-ray RGB loss against each training pixel, whereas the alpha value is determined by the mask loss from an image segmenter. An overview of the method is shown in Figure 2.14. The novelty of this method, lies in the NeRF formulation, where it draws random rays from the entire image training set and associates each latent code, to the ray corresponding to the face image it was

sampled from. Overall, LOLNeRF, achieves state-of-the-art results in novel view-synthesis of faces, as well as reconstructing the 3D structure and predicting the respective depth.

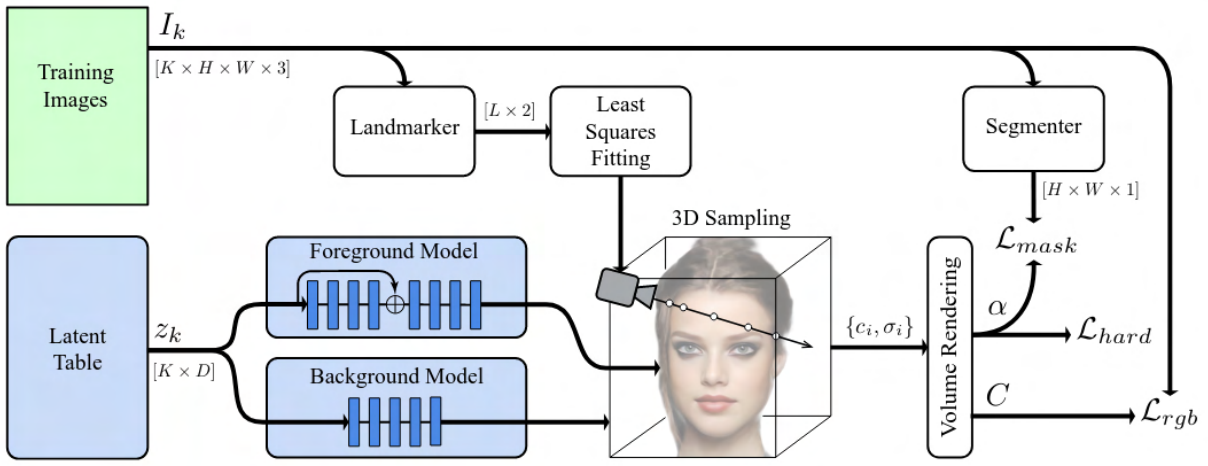


Figure 2.14: LOLNeRF pipeline. The latent code table of K latent codes, corresponds to K training images. The latent code table, along with the Foreground and Background NeRFs are learned through a volume representation, with decomposed foreground and background details. The cameras for the various training images K need to be aligned (least squares fitting of 2D landmarks and 3D canonical keypoints). The space of geometry and appearance fields, are learned optimizing an loss consisting of an rgb, mask and hard loss. Image from [39].

2.1 Depth-Based 3D reconstruction

Training on 3D data is time-consuming, inefficient, which accounts for the 2D training data supervision supremacy, even in the task of 3D reconstruction. Methods, that predict depth face values [57] [33] [19] or normal maps [4] [19], achieve 3D reconstruction, by inferring 3D structure [47] [19] [51].

Unsup3D [57] learns 3D objects directly from single-view images without no other supervision. Given a single image, Unsup3D is able to recover instance-specifc 3D shapes, as well as pose, texture and depth. The key idea is to exploit the bilateral symmetry in these objects. Hence, for face images, there is face symmetry, between the left and right side of face. Training is carried out through framework called, photo-geometric autoencoding, that disentangles, the 3D shape (represented as a depth map), the texture and the camera pose, from a single image. The latter are fed to a renderer, that produced the reconstructed render image and is trained by optimizing the difference between the input and render image, via a reconstruction loss. This decomposition is only possible (avoid degenerate solutions), by adding extra constraints and enforcing symmetry. The model is enforced to predict a symmetrical view of the depth and texture (flipping of each side), by switching and optimizing between 2 reconstructions (for original and flipped version), simultaneously. Finally, to tackle non-symmetric lightning (e.g non symmetric albedo, deformation, etc.), the framework models the uncertainty, by additionally outputting 2 confidence maps and optimizing the 2 ”confidence-adjusted” reconstruction losses.

DepthNet [33] is an unspervised method that infers 3D structure by predicting facial depth, without gt depth map information. It also predicts parameters of 3D affine transformation matrices, to assist in in transforming existing faces to novel face poses. The pipeline leverages neural networks for inferring depth and geometric face transformations (Siamese Network [52]), as well as to improve the appearance of the 3D model, through

image-to-image transformation network (CycleGAN [64]). Predicting parameters for the pose transforation matrix, DepthNet can repose a source face image, to a target pose, by retaining the 3D shape and appearance of the source face image. A NN uses 2 input images (A and B) and their respective 2D keypoint landmarks, as input. 2 input images are fed into a simaese network, that tries to repose the source image (A) to the target pose of image (B) and by doing so it finds a transformation matrix for the predicted depth facial keypoint values. The 2D facial landmarks are concatenated in the extracted feature vector of the Siamese network. In essence, the network is optimized, by minimizing the loss between the 2D predicted keypoints (for reposed source image A) and the 2D facial landmarks of target pose image B. DepthNet use a "novel loss formulation for the structured prediction of keypoint depths" [33]. By estimating the 3D affine transformation matrix, reposing is achieved, by warping a source image onto the target face geometry (via a textured triangular mesh) (Figure 2.15).

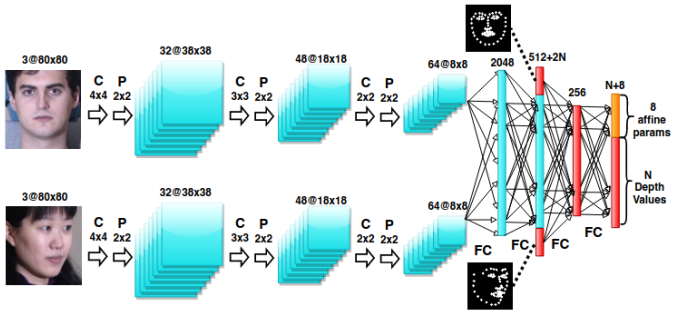


Figure 2.15: DepthNet pipeline. Input 2 images A and B and pass them through a Siamese network. The extracted feature vector, are passed through fully connected layers, and are concatenated with the facial 2D landmarks of each image. Finally depth facial values and parameters of the transformation matrix are predicted. Leveraging this information, predicted reposing 2D facial landmarks are compared with the 2D facial landmarks of the target pose B, in an attempt optimize the network. Figure from [33].

2.2 Regression Based

Several methods, regress directly latent parameters from single input images to infer 3D structure [13][**?**][49]. Specifically, methods like DECA [13], directly learns to regress FLAME parameters, along with other latent parameters to obtain better 3D reconstructions. In similar fashion, **RingNet** [44] leverages an encoder-decoder architecture, to reconstruct 3D geometry of a face. The image encoder extracts face features, whereas the decoder is the FLAME model. In essence, RingNet, takes a single input image and predicts the corresponding FLAME parameters, that lead to the final 3D mesh. During training, it requires multiple images of the same person and a single image of a different person. This way, it learns to regress FLAME parameters, while optimizing a shape consistency loss for same subject images and an inconsistency loss for different subjects (See Figure 2.16).

The key assumption is that images of the same person, always yield the same unique face shape, despite varying lighting conditions, pose etc. The opposite assumption does not stand, for face images of different people/subjects. This is formulated into our network as 3D shape consistency and inconsistency losses. The encoder network (ResNet 50) extracts feature vectors from the image, that is then regressed to appropriate latent FLAME parameters related to pose, expression, illumincation, jaw. The shape consistency (for same subjects) and inconcistency (different subjects) is simultaenously enforced during training, by minimizing the difference between the 2D ground truth face landmarks and the projected 3D landmarks from the output 3D mesh. During inference, the output 3D mesh is obtained by inputting the appropriate regressed parameters in the FLAME model (see Figure 2.16)

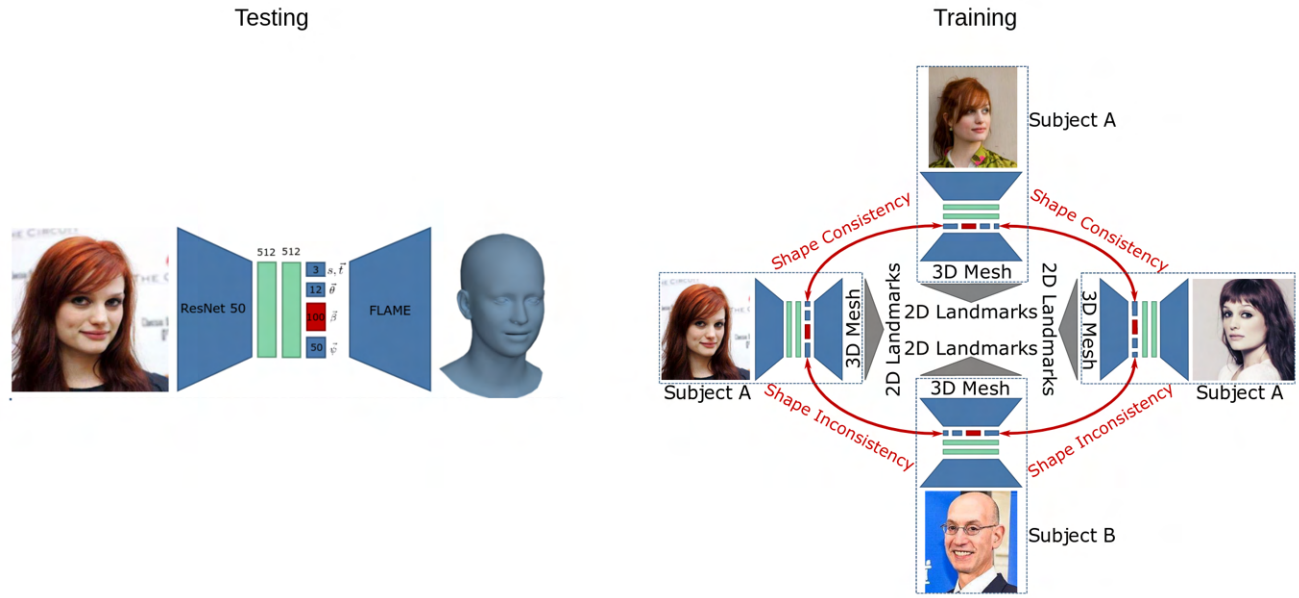


Figure 2.16: RingNet training and testing pipeline. A method that regresses FLAME parameters/coefficients (pose - θ, expression - ψ, shape - β and neck, jaw and eyeballs $(s, \mathbf{t})$, for 3D mesh reconstruction, from image face pixels. It does not require 3D supervision and leverages an image encoder, a regressor and FLAME model as decoder. Figure style adapted from [44].

This method reconstructs face shape without hair and high-frequency facial details, however it still underlines that regression-based methods offer a fast and reliable solution to single-view 3D reconstruction.

Chapter 3

Methods

Our framework, leverages EG3D-generated synthetic data, to train depth and latent code reconstruction losses in a supervised manner. We present 2 seperate frameworks, that predict depth and EG3D latent codes [9], by regressing directly from image face pixels. The 2 methods, solved seperately a depthmap and latent-code-regression problem and are presented in the following sections 3.1, 3.2 respectively. The models are trained on synthetic face data (generated with EG3D data [9]), by leveraging different architectures, that vary for the 2 tasks. For depthmap regression, the MiDaS [36] architecture, along with the Attenion-UNET [42] architecture is employed, to predict the facial depth directly from a single-view RGB face image. For the latent-code task, the UNET [42] architecture, as well as the Vision Transformers (ViT) [54] are used, to regress EG3D intermediate latent codes, from a single-view RGB image of a human face. All models, are trained mainly on frontal-view, tightly cropped face images. The depthmap regression models, achieves 3D reconstruction, by inferring 3D structure from the depth predicted values. On the other hand, as part of the latent-code task, given an input face image, the predicted EG3D latent code is fed in the EG3D generator, to generate the corresponding depthmap, and reconstruct the face's appearance and geometry.

Overall, this section introduces the methods of inferring 3D structure using regression based methods, hence either through depth or by leveraging the EG3D latent code. The 2 main methods, focus on regressing depth directly from the input RGB single images and reconstructing the 3D human face, or by regressing the EG3D latent code, that can reproduce the image input and the equivalent geometry.

3.1 Depthmap regression

3.1.1 Overview

Depth prediction, refers to predicting the extra z coordinate in the world coordinate system from an x, y picture (width, height) in the image plane coordinate system. Given an input image RGB, a model can predict the depth of each pixel in the image - d. The same applies, for images of human faces, whereby leveraging a neural network architecture, depth is estimated. Regressing depth, requires a standard preprocessing step, to feed in a standard procedure, the input image in the network, and predict depth for each pixel.

The method is based on a sophisticated neural network architecture, that is trained on ground truth (GT) depthmaps of the corresponding input face images. The depth regressor $model$ learns a mapping between the input face image I and the dense depth correspondence d. The network is trained on synthetic images generated by the EG3D network pretrained on the FFHQ dataset [24]. EG3D generates face images, and additionally the corresponding depthmap, that is used as the main ground truth for the supervised model training. The data generation process, is described in more detail in the following section 3.2.3 . An overview of pipeline training and testing is presented below as a pseudocode/algorithmic implementation (see Algorithm 1, 3) for both models (sections 3.1.3, 3.1.4).

Algorithm 1 Depthmap regressor: Training

procedure DATA GENERATION VIA EG3D
  $I \leftarrow$ Generate images through EG3D
  $d \leftarrow$ Generate corresponding Depthmap through EG3D
procedure TRAINING PIPELINE
  for every image I **do**
    $I_{pre} \leftarrow preprocessing(I)$ ▷ Preprocessing of Input Image - *I*
    model $\leftarrow$ Define regressor model architecture ▷ MiDaS or UNET
    $d_{pred} \leftarrow model(I_{pre})$ ▷ Predict depthmap
    $loss \leftarrow l1Loss(d, d_{pred})$ ▷ Compute l1 loss distance between gt and predicted depthmap
    Minimize depth reconstruction loss - $loss$ ▷ by backpropagation and updating model parameters

Algorithm 2 Depthmap regressor: Testing

procedure TESTING PIPELINE
  $I \leftarrow$ Input face image ▷ Front view face image
  $I_{pre} \leftarrow preprocessing(I)$ ▷ Preprocessing of Input Image - *I*
  model $\leftarrow$ Define regressor model architecture ▷ MiDaS or UNET
  $d_{pred} \leftarrow model(I_{pre})$ ▷ Predict depthmap
  $mesh \leftarrow depth2mesh(d_{pred})$ ▷ Reconstruct 3D untextured face mesh from depthmap
  $mesh_{tex} \leftarrow fitMesh(mesh, I)$ ▷ Add texture in mesh by analysis-by-synthesis (optional)

The algorithmic implementation constitutes of basic components such as the model architecture ($model$: $I \mapsto d$), the EG3D model used for data generation and other functions that complement the training and testing pipelines. The depthmap training is performed for both model architectures seperately and training, validation and evaluation scores are reported in section 4.3.2.

The pipeline of the depthmap prediction pipeline varies between the training and testing phase, as explained in the respective high-level algorithms (3, 1). A visual representation of the training and testing pipeline is shown in Figures 3.1, 3.2.

During training, an RGB square face image of high resolution, is given, as input. High resolution, refers to an input image of higher resolution than 512x512 pixels. In our method, all training input images, are synethetic and are generated by EG3D. The preprocessing step is implemented in the input synthetic RGB image, to obtain the resized, standardized and normalized input to the the model, of size 128x128 (shape = (128,128,3)). The neural network, regresses depth directly from the input iamge, to predict dense depth values for each pixel. The output depthmap is of size (128,128) and is compared to the ground truth depth map of size (128x128), generated by EG3D. The model updates the weights and optimizes an l1-loss for each depthmap pixel. By minimizing the depth loss between the predicted and ground truth depth map,the model learns to predcit dense depth values for any face image, inferring an accurate 3D structure of the human face, directly from an RGB image. EG3D is essential to this method, as it provides a vast source of synthetic training data ($\sim 50,000$ images), as well as the corresponding depthmaps.

During the testing stage, similarly to the training stage, an RGB image of a human face is given as input. Preferably, the image is of high-resolution, to preserve the high-frequency details and salient features of the human face, and allow the model to achieve better results. The preprocessed RGB input image (128x128), is fed to the model and dense depth is predicted for all image pixels. The predicted depth, refers to just relative depth, which serves the purpose of our task. Through the "depth2mesh" function, the depth values that infer 3D structure, are transformed to a 3D mesh. This generated triangular mesh, consists of a high number of vertices, and their respective faces, which are formed, by connecting 3 vertices, dictated by 3 vertex indices. The triangular mesh is untextured, meaning that there is no assigned RGB value to the faces of the mesh, but

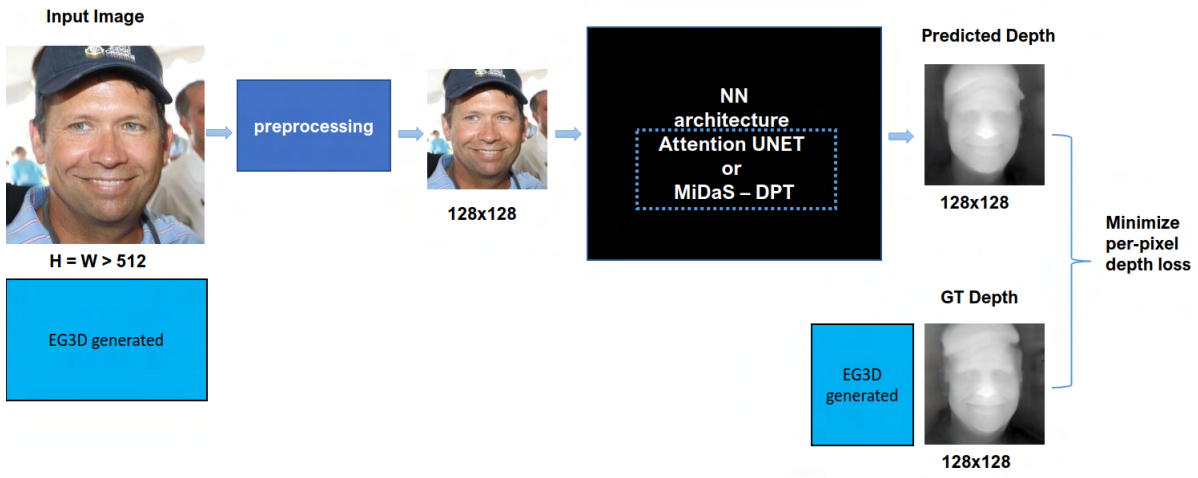


Figure 3.1: Training pipeline of depth regression task. An EG3D synthetic input image is preprocessed accordingly to the a normalized,standardized image of appropriate model size. The model accepts input RGB images of height and width 128x128 pixels. Neural network (NN) architecture selection, is between the described Attention UNET and MiDaS-DPT models. The model, predicts dense depth predictions, by optimizing a depth reconstruction loss (l1-perpixel loss), between the predicted and ground truth depthmap (initially generated form EG3D).

only geometry is reconstructed. To further reconstruct the texturing, we can fit the aforementioned mesh. This implies, that an initialized texture UV map, is progressively optimized, in an analysis-by-synthesis fashion. By optimizing the UV texture map, as well as the camera pose parameters, the mesh is rendered and compared to the input target image. BY minimizing the image l1-losses, an appropriate UV texture map is generated, that can be directly layed upon the generated geometry 3D mesh. Reconstructing the texture, is optional and is not supervised during training, as an end-to-end differentiable method. More details about fitting the mesh and extracting the UV texture map, -to finally generate a textured 3D human face mesh- can be found in section 3.1.6.

3.1.2 Preprocessing

For the depthmap regression task, for any of the 2 architectures, the same preprocessing step is applied. The required input, is an RGB image of size 3 x 128 x 128, where the first dimension refers to the number of channels, and the last 2 dimensions refer to height and width respectively. The input image is also normalized between 0 and 1. The model, in general performs better in frontal-view- and tightly cropped- faces. The standard preprocessing pipeline, borrows from the input preprocesing step of MiDaS [36] and involves a composition of image transformations. Assumming a face RGB image of random size - shape = (width, height, 3), preprocessing steps involve:

1. Resize input image to appropriate height and width (128, 128, 3)
2. Normalize input image to float values between 0 and 1
3. Standardize input image to mean 0.5 and standard deviation 0.5
4. Normalize input image to float values between -1 and 1
5. Standardize input image to mean 0 and standard deviation 1

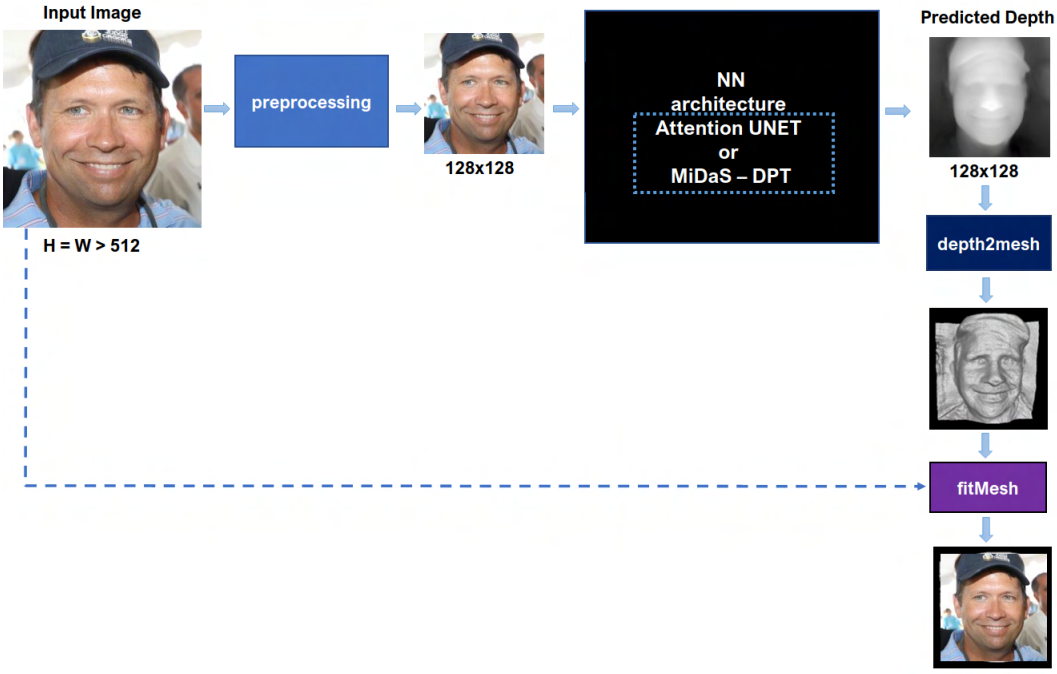


Figure 3.2: Testing pipeline of depth regression task. An square image of random high resolution (H and W > 512 pixels), is preprossessed before being fed as input to the neural network (NN). H and W stand for image height and width.The model, inputs a 128x128 2D image (RGB) and predicts the dense depth values. The output depth is of size (128,128). The 3D geometry of the input face image is reconstructed, by applying the "depth2mesh" function, to the output depthmap. The reconstructed mesh can be optionally textured, by fitting the geometry mesh ("fitMesh" function).

6. Change dimension shape, to match appropriate model input size (3, 128, 128)

For better reconstruction performance, a face detector can be added as a preprocessing step. By identifying the face region and cropping tightly around a face (as a bounding box), the input is closer to the training distribution. In other words, the input is closer to the input expected from the model. This can be achieved by implementing a lightweight fast face detector, such as MTCNN [60], to enable reasonable inference times, or real-time model deployment.

3.1.3 Attention - UNET

The framework is implemented by leveraging the Attention - UNET architecture [34], that built on top of the original UNET formulation [42], with the addition of an Attention Gate (AG) unit. The architecture is modified also, in comparison to the original UNET, nevertheless, the main architecture and idea follows in the steps of UNET. The architecture consists of 4 encoder and 3 decoder blocks, that define the expansive and contracting path. The skip connections propagating features from the encoding blocks are passed through the novel AG, to achieve feature selectivity and filtering. The architecture is based on 3D convolutions, to account also for 5D input tensors, often required for 3D medical data (initial application of UNET).

An overview of the Attention - UNET architecture can be shown in Figure 3.3. Similar, to the UNET architecture, the Attention- UNET, has a U-shape architecture consisting of a contracting path and an expansive path. The contracting path, consists of a series of encoding blocks that extract feature maps, by downsampling them by a factor of 2, whilst doubling the number of feature channels (filters) at each step. Each encoder block, extracts features from the input image, by passing the input image through 2 convolutional layers of

kernel size 3x3x3, followed by a ReLU activation function each. The ReLU output, acts as a skip connection for the respective decoder block, whereas max pooling of size 2x2x2, is performed and the output is fed to the next encoder block. ReLU is widely used in deep learning to introduce non-linearity into the network, to improve the complexity of the network, whereas the max pooling layer, is used to aggregate the feature map and reduce the number of trainable parameters. The use of max pooling, instead of adaptive pooling, allows also dense pixel predictions, which is essential for predicting per pixel depth. Skip connections, allow features from earlier layers, to be used by the decoder blocks, to achieve better overall results. In the original formulation, the propagated features are concatenated, directly with the decoding block features, however the AG units are applied to the propagated features, to identify only relevant-to the task-features. Skip connections, provide additional feature information to the decoder network, mainly salient features (e.g edges of the face) that can help the decoder achieve a more precise localization. They act as a shortcut connection, that allows the indirect flow of gradients to earlier layers during backpropagation and hence results to better overall representation.

The output of the last encoding block, results to a feature map with spatial dimensions 8 times, smaller than the original image. The output is then upsampled, in each step of the expansive path. The encoding block output, is upsampled and concatenated with the skip connection features, passed through the AG. AG filters the features from the skip connections

This attention gate model, combines self attention, with grid-based gating, that allows the attention coefficients to better localize features. This is a soft-attention gating technique (weighting different parts of the image), that endorses gating (use of contextual information) based on a more locally-oriented vector. In other words, it generates a gating signal that is "end-to-end trainable" (differentiable), and allows the network to contextualise local information useful for prediction" [45]. On the one hand, soft attention, actively suppresses activation at irrelevant regions, whereas the gating function, combines the contextual features, with higher-level features. Overall, AG, receives 2 inputs, a gate signal (query) - g and an input signal x, corresponding to the decoding block features and the skip connection features respectively. The gate signal coming from the skip connection has better spatial information, whereas the input signal, comes from deeper layers of the model and hence has better feature representation. The input and gating signal are added, before being passed through a ReLU activation σ_1. Then, the activation output is passes through a 1x1 convolution ψ, that scales the weights via sigmoid - σ_2. The weights are then upsampled - a, to match the original size of x signal. Then the attention weights are multiplied element-wise, with the input signal x, to retrieve the output signal x_l. The schematic of the AG unit, is shown in Figure 3.3.

AG units improve the selectivity of features, and force them to be more specific to local regions (e.g the face foreground and not the background). The expansive path, progresses by upsampling the feature map's spatial dimension and halving the number of feature map channels.

The features are passed, are passed through a series of 3 decoder networks, as part of the expansive path, during which feature map upsampling is achieved, by doubling the feature map spatial dimension and halving the number of feature map channels(number of filters). Each decoder block, inputs feature maps, that correspond to an abstract representation of the face, and generates a feature map that corresponds to more detail face representation with improved semantic details. Specifically, each decoder block, consists of a 2x2x2 transpose convolution ("up-convolution"), followed by a concatenation of the output features, with the skip connection features (passed thruogh the AG unit). The concatenated feature map is fed to 2 additional 3x3x3 convolution, followed again by a ReLu activation function. The output of the original Attention - UNET is a segmentation mask, with N_c referring to the number of classification/segmentation classes.

Our architecture, is the same as the original Attention - UNET architecture with some additional modifications, to account for predicting pixel depth values, instead of multiple class probabilties, as well as working on 2D input images (4D tensors). The main modifications are:

- 2D input image data is used (instead of 3D scans), meaning the model expected input is (N, C, H, W), where N is the number of samples images, C is the number of channels (C = 3) and H and W refer to image height and width (H = 128, W= 128)

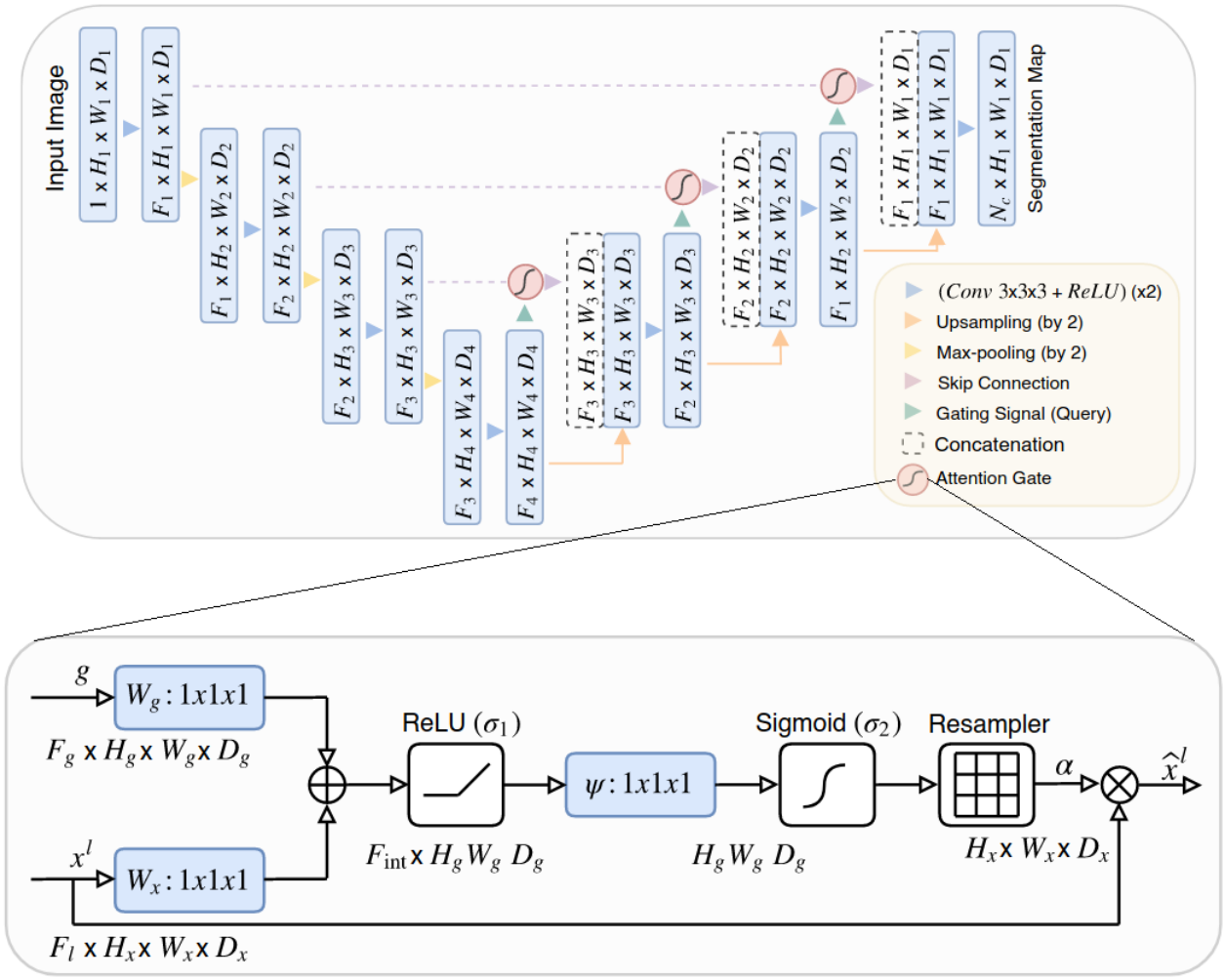


Figure 3.3: Attention UNET architecture with Attention Gate (AG) unit. The architecture consists of a contracting path (left side) and an expansive path (right side). The contracting path, follows a series of feature map downsampling, with double number of filters at each downsampling step. The expansive path, involves a series of upsampling steps, with additional feature information form the skip connections. Each upsampling step, involves an "up-convolution", with the number of spatial feature map dimensions doubling. The features from the skip connections, are not cropped , as padding is used in 3x3x3 convolutions, to prevent loss of border pixels. N_c denotes the number of classes (applies in classification/segmentation tasks), whereas each AG filters the features propagated by the skip connections. F refers to the number of filters (or features), H and W, to image height and width respectively and D to the channel dimension (e.g for RGB image, D=3). The Attention Gate unit (AG) receives an input feature map x_l and a gating signal g collected at a coarse scale. The blue boxes in the AG, are linear transformations implemented as 1x1x1 convolutions, to successfully find the spatial attention weights α, to map to the output signal x_l. Image adapted from [34].

- Number of encoding blocks are 5 (increased depth in expansive path), to capture more contextual information
- Number of decoding blocks are 4 (increased depth in contracting path), to generate better feature representation
- The up-sampling method, is a resize convolution with bi-linear interpolation, instead of an "upconvolu-

tion" (transpose convolution)

- Batch Normalization is applied to every image batch after the end of each block
- Output is of size (N, C, H, W), where N is the batch size (or number of samples), H and W are the image's height and width (H=128, W=128), C is the number channels, which is 1, in this case as we are just predicting dense depth values
- The model output for a single image is a (128,128) image with depth values

3.1.4 MiDaS - DPT

Another model used for estimating face depth from a single-view image, is MiDaS [36]. MiDaS heavily relies on pretrained image encoders, trained on auxiliary tasks. MiDaS uses some mixing dataset strategies, to improve depth performance. This means that, the model is trained on 10 diverse datasets (initial paper used 5). The training datasets, have a wide source of different data, with different ground truth. The main source, is 3D films (not 2D animation), web videos, and "in-the-wild" images from the Internet. The different data sources, have different depth ground truth, such as metric depth, relative depth, disparity (inverse depth up to scale). Hence a multi-objective is used, to account for the various data sources and inverse depth maps are used as ground truth. A robust training objective based on inverse depth is learned, scale and shift invariant (invariant to changes in depth range and scale). MiDaS is used for zero-cross dataset transfer and can be directly used in any unseen domain for depth prediction(such as faces). Nevertheless, as part of our training pipeline, we leverage the model architecture used in MiDaS. Initially MiDaS, used CNNs as the main backbone, specifically ResNet networks [18]. Even though, is performs well, more recent advancements in computer vision, such as Vision Transformers [54], provided a better alternative for dense prediction tasks. As a result, the MiDaS backbone, is an architecture termed as DPT-Large [37] and offers a robust solution to dense depth prediction using vision transformers [54]. The model architecture for the depth regression task, is the DPT-Large model, that consists of 24 transformer layers, with a 16x16 patch embeddings as input. The architecture is visualized in Figure 3.4 below.

DPT-Large uses as encoder backbone ViT-Large with 16x16 local patch embeddings, with feature dimension (after linear projection) of D = 1024 features. Leveraging a Vision Transformer as the backbone, means that the input image is divided to 16x16 non-overlapping local patches. Each patch is flattened and linearly projected, to an embedding space. The obtained patch embeddings are augmented with some position embeddings. In contrast, to the traditional approach, DPT-Large, uses an extra special token, called "readout" token, which does not depend on the input image. This can later be used, as a global image representation, assisting in the task of dense depth prediction. All aforementioned tokens, are fed in the various transformer layers, where through Multi-Head Self Attention, it learns a more global representation of the image, by building the entire receptive field throughout only a single layer. The output of some transformer layers (layers = 5, 12, 18, 24), is fed to a "Reassemble" block. The purpose of this block is to "read" (handle) the "readout" token, use spacial concatenation of the tokens, to reshape the tokens to an-image like representation and upsample the feature map spatial dimension, through a spatial resampling layer. In more detail, during the "read" step, the "readout" token passes inherently its information to the remaining tokens through projecting the concatenation (of all tokens and the "readout" token) to the original feature dimension space D [37]. The tokens are then spatial concatenated, to a feature map representation of size $\frac{H}{16} \times \frac{W}{16}$ with D channels, where H and W correspond to input image's height and width respectively. The last step is up-sampling this feature map representation to a size of $\frac{H}{s} \times \frac{W}{s}$, where s is 4, 6, 8, 16 or 32, with $\hat{D} = 256$ feature channels. In general deeper layers have lower resolution feature maps, and vice verca. These feature maps with various spatial dimensions are then propagated to "fusion" blocks, that progressively builds and upsamples (by factor of 2 at each fusion stage) the

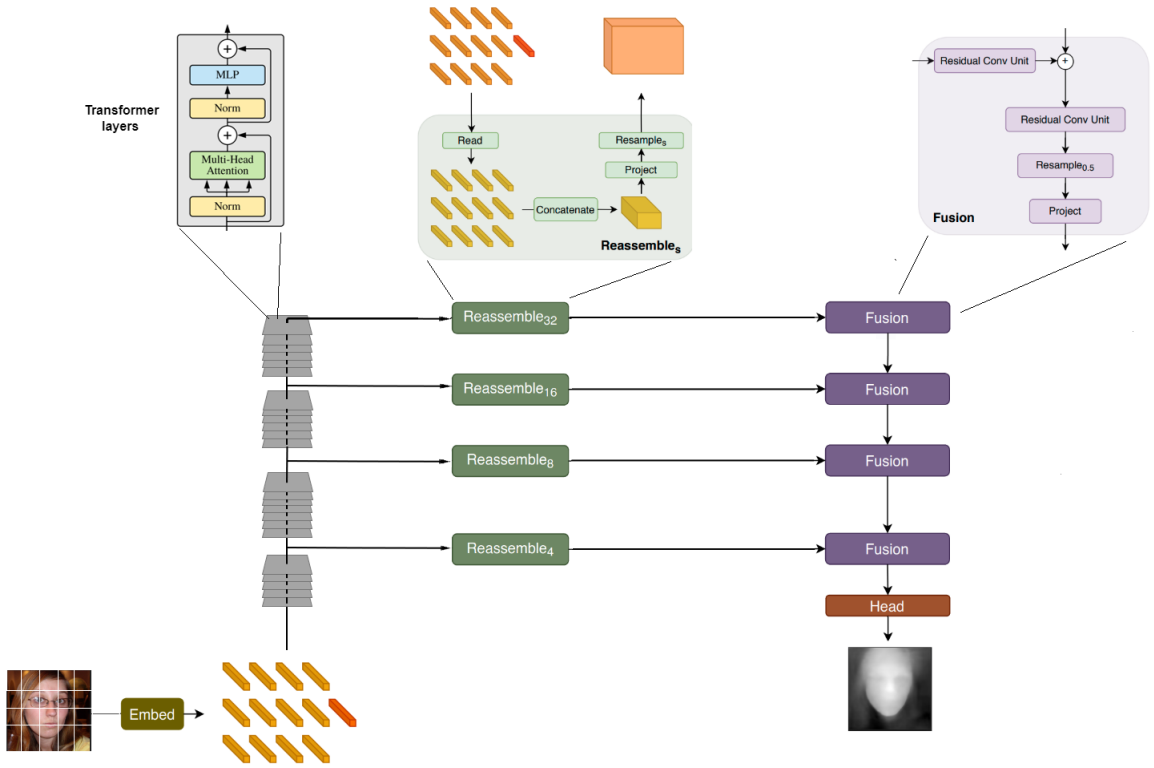


Figure 3.4: DPT-Large architecture: Image is embedded to tokens, and augmented with a positional embedding (orange). The red token, is an additional special token, termed as "readout" token. All these tokens are parsed as inputs to a series of transformer layers (grey) (24 layers for DPT-Large), that leverage Multi-Head Self Attention. The output from transformer layers (5, 12, 18, 24) is propagated through a reassemble module (green), that assembles a set of tokens to image-like feature representations with spatial dimensions - s times smaller than the original image size. These feature representations are then progressively fused to the final dense prediction, via the "Fusion" blocks (purple). DPT-Large architecture is a classic encoder-decoder like architecture, with transformer layers acting as the encoder, and "Reassemble" and "Fusion" blocks as a convolution decoder.

feature map, to obtain a feature map of size $\frac{H}{2} \times \frac{W}{2}$, and approximately align with the image representation for dense depth prediction.

3.1.5 3D reconstruction from depthmap

All components above are detrimental to the implementation of the model pipeline and training. However,once the model is trained, the main goal in the inference stage is to solely predict depth and infer 3D information from the input image. In essence, the an accurate enough depthmap of the human face, results to a high-quality 3D face reconstruction. The following algorithm/Figure demonstrates the full pipeline, during the inference stage. After the network predicts the depth of the image, an external algorithm is implemented, that reconstructs a 3D mesh from the depth [47][1]. The algorithm is summarized below and it's a standard method for recovering the vertices, faces and edges of a polygonal mesh.

Algorithm 3 depth2mesh - Mesh generation from depth

procedure DEPTH2MESH(d_{pred})
  $H \leftarrow$ depthmap d_{pred} height
  $W \leftarrow$ depthmap d_{pred} width
  procedure GET VERTICES
    for every depthmap pixel **do**
      $x_i \leftarrow$ pixel width value ▷ scaled by W
      $y_i \leftarrow$ pixel height value ▷ scaled by H
      $z_i \leftarrow$ pixel depth value ▷ relative depth
      $v_i \leftarrow (x, y, z)$ ▷ define the mesh vertex
      $v \leftarrow$ Store all v_i ▷ store all vertices for all pixels
  procedure GET FACES
    for every depthmap pixel **do**:
      $local \leftarrow$ depth pixel values of local region ▷ extract local patch with depth values
      $res \leftarrow$ calculate depth difference for adjacent pixels in $local$ ▷ absolute difference
      $threshold \leftarrow 0.1$ ▷ accepted depth difference between local pixels
      if $res < threshold$ **then** ▷ depth value residuals for local pixels should be within limit
        $v_i \leftarrow$ vertex corresponding to current pixel
        $v_r \leftarrow$ vertex corr. to right pixel of current pixel
        $v_r \leftarrow$ vertex corr. to right pixel of current pixel
        $v_d \leftarrow$ vertex corr. to pixel below current pixel
        $v_d r \leftarrow$ vertex corr. to down and right pixel of current pixel
        $f1 \leftarrow (\mathrm{v}_i, v_r, v_d)$ ▷ face formed by connecting these vertices
        $f2 \leftarrow (\mathrm{v}_d, v_d r, d_r)$ ▷ face formed from these vertices
        $f \leftarrow f1$ and $f2$ ▷ store all faces for all pixels
  $m \leftarrow$ mesh defined by v and f ▷ Define mesh by connecting vertices and forming faces

The algorithm operates also in face triangles, and extracts the triangular mesh from the depth values. The extracted mesh, represents the 3D geometry of the face, and has no specific RGB color assigned to the mesh vertices, as there is no texture representation. The algorithm simply describes a non-textured 3D reconstruction of a human face, from a depth via triangular mesh generation. In case, the output depthmap has also background information, the vertices belonging in the foreground face are only filtered in, via a mask.

3.1.6 3D Mesh texture

Once the face geometry is reconstructed, a textured version of a 3D human face can be also be obtained. This can be implemented optionally, as part of the testing pipeline. The reconstructed 3D geometry, is a 3D mesh consisting of N vertices, and F faces, that correspond to vertex positions that form the mesh faces. Each vertex has 3 coordinates, as it holds an x,y,z value, so it is defined in the 3D space. Each face F, has 3 indices of any of

the N vertices, that are connected to form the face. This is a simple explanation of an untextured 3D triangular mesh, that is generated from the "depth2mesh" function. To define a textured triangular mesh, usually an RGB color, assigned to each vertex or a texture UV map is defined. Texture represented as a color 2D UV map is later on layed on the mesh and rendered accordingly. To obtain a textured 3D human face mesh, the method of fitting a mesh is followed. In more detail, we initially define a renderer, by providing the camera pose (extrinsic and intrinsic parameters), as well as defining the mesh rasterizer, the lighting (PointLights) and the appropriate shader (SoftPhongShader) [38]. In the implementation side, this is achieved by using a fully differentiable renderer from Pytorch3D [38]. Once the renderer is setup, the texture UV map is initialized. This can be initialized with any random RGB color (e.g all black). As a next step, the reconstructed mesh, is rendered along with the texture. The final render is compared to the input image and a pixel-wise l1-loss is computed. Hence, by optimizing the camera parameters of the renderer and the texture UV map, the generated rendered textured mesh is also optimized, to minimize the pixel difference between the final render and the input image. Simply stated, by changing the viewing angle and texture color, we change the final render, so it looks as similar as possible, to the input image. This optimization runs for a number of epochs , leading to the final optimal UV texture map, leveraged for textured 3D mesh reconstruction.

The algorithmic high-level process of fitting a mesh, is presented in Algorithm 4, where the generated 3D geometry mesh - m (from "depth2mesh"), as well as the input face image - I, are given as function inputs.

Algorithm 4 fitMesh

procedure FITMESH(m, I)
cam $\leftarrow$ extrinsic and intrinsic params ▷ Define Rotation, translation, focal length, optical centre
$shader \leftarrow$ soft phong shader ▷ Define shader with Point lights
$rast \leftarrow$ mesh rasterization ▷ Initialize rasterizer
$r \leftarrow render(cam, shader, rast)$ ▷ setup renderer
$tex \leftarrow$ UV black color ▷ Initialize UV texture map
for every epoch **do**
$I_r \leftarrow r(m, tex, cam)$ ▷ Render textured mesh
$loss \leftarrow l1Loss(I, I_r)$ ▷ Compute l1 loss pixel distance for render and input image
Minimize $loss$ by updating tex and cam ▷ Wrap an optimizer around these parameters ▷ and backpropagate

According to the algorithm, above, the camera pose parameters, as well as the texture UV map are updated in every epoch, in an attempt to minimize the pixel distance between the final render and the input image. This is achieved by creating an optimizer object around this parameters, and by training via SGD, the optimizer model.

3.2 EG3D latent code regression

3.2.1 Overview

An alternative method, for obtaining a 3D face mesh from a single input image, is to leverage directly the EG3D pipeline. EG3D is a geometry aware powerful generative model, that is pretrained on FFHQ, specifically on human faces. As a result it can not only, generate synthetic images, but also depthmap and the 3D shape of the face, directly from its implicit rendering pipeline. EG3D requires camera parameters as input, as well as a sample from its latent space. The latent code is hence fed with the camera input in the generator and the image (along with its constituent outputs) is synthesized. The ability of EG3D to generate 3D aware images and their corresponding shapes, given a specific latent code, is quite useful. Assume that we knew the corresponding latent code, for a single input image. This would mean, that by parsing in the latent code, along with fixed camera parameters, the initial input image could be reproduced, along with the corresponding 3D shape. This

can be achieved by inverting an image to the pivot latent code, in essence in a latent representation in the EG3D domain. However, this method, need retraining of more networks (projection network and generator finetuning) and hence is time-expensive. To counter this problem, we propose a EG3D latent code regression pipeline, by directly predicting the corresponding EG3D ppivot latent code, directly from image pixels. Given, a single view face image, the regression model should obtain the latent code, that if fed in the EG3D generator (along with the camera pose parameters), the exact input image is synthesized.The are mainly 2 regression model architectures presented in this work. The first neural network architecture, is based on a modified version of U-NET following an encoder-decoder structure, to leverage CNNs and produce good feature represenations. On the other hand, a novel model architecture, called Vision Transformers (ViT) [54], that extracts features and builds the entire receptive field within a single layer pass, is another architecture alternative for the latent code regression task. Directly regressing the EG3D latent code from a single-view image, means that the 3D shape of the target face is reconstructed, along the EG3D pipeline.

During training (see Algorithm 5), a regressor model R learns a mapping between an face image I and the EG3D latent code w. The regressor model, is trained on ground truth latent codes, that have been generated from EG3D, along with their synthetic image. This can be thought as an GAN inversion procedure, where $R : I \mapsto w$, instead of generating an image based on a sampled latent code, which is in essence the main principle behind generative models.

Algorithm 5 EG3D latent code regressor: Training

procedure DATA GENERATION VIA EG3D
  $I_{gt} \leftarrow$ Generate images through EG3D
  $w_{gt} \leftarrow$ Store corresponding EG3D intermediate latent code
procedure TRAINING PIPELINE
  for every image I **do**
    $I_{pre} \leftarrow preprocessing(I)$ ▷ Preprocessing of Input Image - *I*
    R $\leftarrow$ Define regressor model architecture ▷ ViT or UNET
    $w_{pred} \leftarrow R(I_{pre})$ ▷ Predict style vector (EG3D latent code)
    $loss \leftarrow l1Loss(w, w_{pred})$ ▷ Compute distance between gt and predicted style vector (latent code)
    Minimize code reconstruction loss - $loss$ ▷ by backpropagation and updating model parameters

After, the model is trained, the EG3D generator is leveraged during the testing stage (Algorithm 6). As a result once the EG3D latent code prediction is obtained , the code is fed to the generator, along with some fixed camera pose and EG3D generates, the 3D mesh of the human face mapped to the respective latent code.

Algorithm 6 EG3D latent code regressor: Testing

procedure TESTING PIPELINE
  $I \leftarrow$ Input face image ▷ Front view face image
  $I_{pre} \leftarrow preprocessing(I)$ ▷ Preprocessing of Input Image - *I*
  R $\leftarrow$ Define regressor model architecture ▷ ViT or UNET
  $w_{pred} \leftarrow R(I_{pre})$ ▷ Predict style vector (EG3D latent code)
  $cam \leftarrow$ Initialize or input camera pose parameters
  $I_{syn}, d_{syn}, vol \leftarrow generator(w_{pred}, cam)$ ▷ EG3D synthesized image, depthmap and voxel grid
  $mesh \leftarrow marchingCubes(vol)$ ▷ convert an implicitly defined volume to a mesh

The training and testing process for the EG3D latent code regression task, is visualized in Figures 3.6, 3.5. The regressor is trained by optimizing an l1-loss between intermediate latent codes w, w_gt of size (14, 512). The predicted latent code w is compared to the ground truth w_{pred}, which was initially stored for corresponding images generated with the EG3D model. Training the latent code regressor, is achieved in a supervised manner, similar to the depth regressor, however in this setting, for 14 intermediate style vectors (size 512), concatenated

together, after the z random latent vector is fed through the mapping network. As observed, the model is not trained on the random z vector of a Gaussian distribution, rather than the intermediate latent code of EG3D (output of mapping network), as it has a distribution closer to the face data. More details can be found in the detailed following sections.

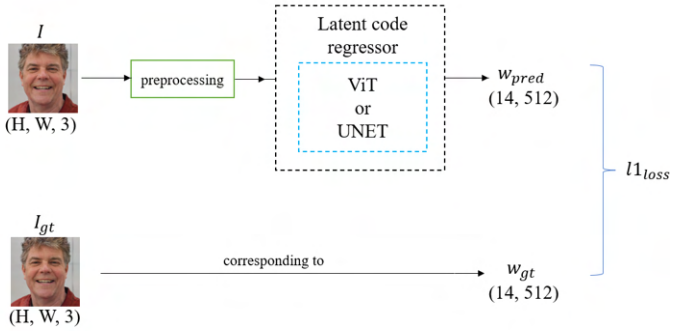


Figure 3.5: Training of the latent code regressor. An RGB face image I is fed as input to the regressor that predicts an intermediate latent code w_{pred}, combatible with the EG3D pipeline. The regressor is trained by optimizing an $l1_{loss}$ between the predicted and ground truth codes w_{pred} and w_{gt} respectively.

Once, the regressor is trained, for any given image I, the model can predict an intermediate latent code w_{pred}, in essence an array with each row representing an intermediate latent style vector. The various style vectors, are then leveraged by EG3D to synthesize the image or depthmap. During the testing stage this block of style vectors - w_{pred} is fed in the EG3D generator, to synthesize the face image I_{syn}, the corresponding depthmap d_{syn} and respective face geometry $mesh$, obtained from the 3D face voxel representation, conditioned on a camera pose - cam. The triangular mesh of the face geomertyu is retrieved by applying the Marching Cubes Algorithm [30] in the 3D voxel grid implicitly defined by EG3D. This is simple algorithm by Lorensen and Cline, that extracts a triangular mesh from an implicit function (scalar field), by iterating ("marching") uniform grid cubes over the volume/object region. Each block of the pipeline for both training and testing stages is detailed in the sections below.

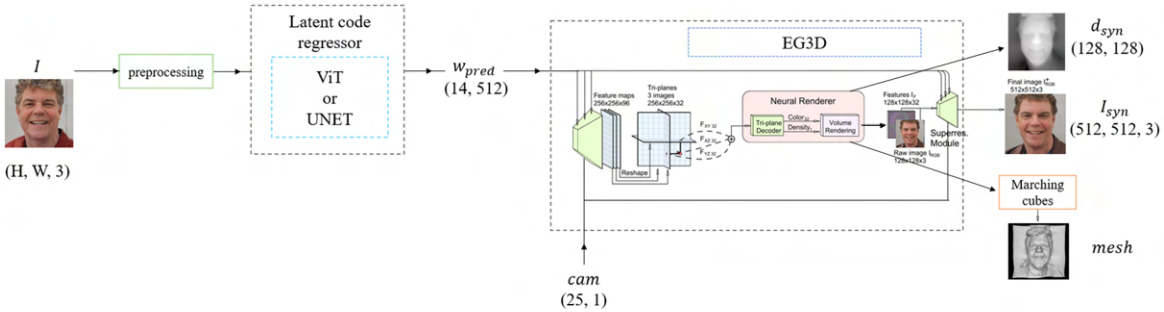


Figure 3.6: Testing pipeline of code regression task. The trained regressor, is fed in with an input image I and predicts the corresponding intermediate latent code w_{pred}. The code is fed to the EG3D generator, directly, by skipping the mapping network. Then by leveraging the tri-plane EG3D generator, and the super resolution module, the face image I_{syn} is generated, along with the corresponding d_{syn} face depthmap and $mesh$ of the face geometry. Figure is inspired by [9].

3.2.2 Preprocessing

For the latent code regression task, for any of 2 models, the same preprocessing step is applied. The required input, is an RGB image of size 3 x 512 x 512 (for UNET) , where the first dimension refers to the number of channels, and the last 2 dimensions refer to height and width respectively. For the ViT architecture the input, is an RGB image of size 3 x 384 x 384. The input image is also normalized between -1 and 1, standardized with mean 0 and standard deviation 1. The model, requires frontal-view- and tightly cropped- faces. The standard preprocessing pipeline, borrows from the input preprocessing step of MiDaS [36] and involves a composition of image transformations. Assumming a face RGB image of random size - shape = (width, height, 3), preprocessing steps involve:

1. Resize input image to appropriate height and width (3, 512, 512) - for UNET or (3, 384, 384) for ViT
2. Normalize input image to float values between 0 and 1
3. Standardize input image to mean 0.5 and standard deviation 0.5
4. Normalize input image to float values between -1 and 1
5. Standardize input image to mean 0 and standard deviation 1
6. Change dimension shape, to match appropriate model input size (3, 512, 512) - for UNET or (3, 384, 384) for ViT

For better reconstruction performance and real-time deployment, similarly to the depthmap prediction task, a face detector can be added as a preprocessing step [60].

3.2.3 EG3D

The latent code regression model, predicts a latent code that is defined in the EG3D domain space. EG3D is a powerful geometry aware generative model, with strong stylizing, and synthesizing capabilities. The importance of EG3D is dual for our framework, as it is used for:

- Generating training data (e.g depthmap, latent code)
- 3D geometry/shape/mesh and appearance reconstruction pipeline

This section, presents a detailed explanation of the EG3D model, as well as its constituent components.

EG3D is a geometrically-aware 3D GAN, that leverages a novel ”tri-plane representation” that combines the best of an explicit and implicit scene represenation. The method builds on existing novel ideas, such as NeRFs [32], voxel approaches(explicit) [29] and the mapping network idea initially introduced by [24]. This 3D GAN produces novel scenes of high-quality human faces, multi-view consistent renderings and the respective detailed geometry [9]. EG3D is trained directly on 2D ”in-the-wild” face images with known camera pose, as there is no need for multi-view supervision or 3D ground truth scans. An overview of the EG3D framework is shown in Figure 3.7. As observed, EG3D uses the main principle and components used in the StyleGAN papers [25][23][24], combined with an explicit-implicit volume representation method. The StyleGAN2 generator is the main generator model for producing high-quality images, that is pose conditioned. Furthermore, a mapping network (similar to [24]) is used to map a sampled latent code to a intermediate latent code, that is conditioned to the camera parameters (extrinsic and intrinsic parameters that define the camera pose). This intermediate latent code represents better the data distribution (of faces), than the Gaussian latent space. A deep dive in the 3D scene ”tri-plane representation” is required, to fully understand the EG3D pipeline.EG3D requires an expressive as well as an efficient 3D representation, to be implemented as part of a generator discriminator architecture for 3D aware face synthesis.

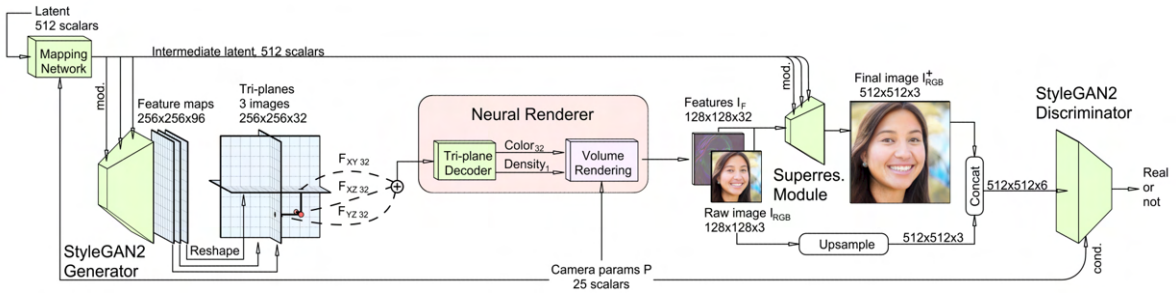


Figure 3.7: EG3D framework overview. The main pipeline components involve a pose conidtioned StyleGAN2 feature generator, along with its mapping network to produce an intermediate latent noise w. The framework enables 3D novel synthesis with the expressive power of a lightweight and efficient tri-plane represenation, followed by a feature decoder. This is part of the neural volume renderer, that is conditioned to the camera pose. The pipeline is finalized with a super resolution module and a pose conditioned StyleGAN2 discriminator that implements the dual discrimination. Figure adapted from [9]

It is is a hybrid amalgamation of an implicit and explicit scene representation method, bearing the advantages of both. Neural implicit scene representations (like NeRF [32]), are not memory expensive, compared to explicit voxel methods [29] (see Figure 3.8), and can scale well with higher resolutions. Neural implicit representations like NeRF leverage fully-connected layers and positional encodings, however it is not very efficient as they are slower to query in comparison to voxel-based methods. On the other hand, explicit methods use smaller implicit decoders (only 2 FC layers) that are faster to query. Nevertheless, an entire voxel grid is built for the entire scene and hence it scales poorly with resolution and becomes computationally heavy. The aim of the tri-plane representation is achieve a less memory expensive 3D scene representation, with fast inference time, but retaining the same or better expressiveness of the aforementioned methods. Instead of storing every single volume density and RGB color of the scene, they have just 3 axis-aligned orthoginal feature planes, that still retrain the depth dimension of the scene. Along each plane (256 x 256 spatial dimension), a feature dimension of 32 is preserved. The number of features are much less than the explicit voxel grid approach, while being more efficient timewise, as during inference we query only a very shallow MPL (compared to the NeRF approach). Specifically, any 3D position is queried by projecting it "onto each of the 3 feature planes" [9], obtaining the respective Fxy, Fxz and Fyz feature vector. These feature vectors are interpreted as color and density via the decoder network (shallow MLP). OVerall, this representation achieves the same expressiveness, as the other 2 methods according to [9]. The trainable weights for the model, are the weights contained in the planes, as well as the weights in the FC layers. The triplane model is trained similarly to NeRF, through "overfitting to a single scene" method.

So according to the EG3D pipeline, some intermediate latent scalars (14 style vectors) are fed to various modules of the pipeline along different pipeline stages. The StyleGAN2 generator is fed with some latent style vectors to generate a feature map of size (256 x 256 x 96), by implementing up-sampling in every layer through some deconvolutions [25]. The feature map is split into 3 planes (256 x 256 spatial dimension) with 32 features to form the tri-plane representation. The hybrid representation is queried and outputs a scalar density σ along with a 3D feature (128x128x32) (32-channels), which is then rendered into a 2D feature image through volumetric rendering (96 total depth samples per ray). As observed, camera parameters are used to condition the mapping network, which leads to pose-correlated feature decoupling (e.g people smile statistically more in front-view images) by the generator. The rendered features along with the intermediate latent style vectors, are fed to the super resolution module [58]. The output of the module is a high-resolution RGB image of the face ${I_{RGB}}^{+}$ of size (512x512x3), which is concatenated with the upsampled blurred version of RGB image I_{RGB} termed as ${I_{RGB}}^{'}$. These 6 channel images are fed to the discriminator, instead of a 3 channel image, leading to this "dual discrimination". Dual discimination, motivates the output images to match the real data distribution,

similar to any other discriminator, with additionally : a) matching the neural rendering with the distribution of down-sampled images b) enforcing consistency between the render output and the super-resolved images [32]. Overall, dual discrimination, tackles the multi-view inconsistencies occurring in 3D GANs.

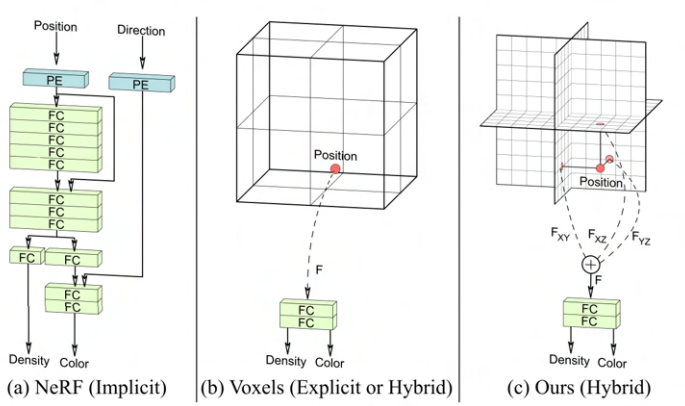


Figure 3.8: Motivation for the hybrid triplane representation (see c). An implicit 3D scene representation leverages fully-connected layers (FC) and positional encoding (PE), . An explicit 3D scene representation is more efficient during prediction, however very memory expensive and does not scale well with high resolution, as an entire 3D volume grid is built and rendered (see b). A hybrid tri-plane representation offers the efficiency of voxel-based methods and scalability of implicit 3D scene representations, with better expressivity (see c). Figure taken from [9].

EG3D is leveraged for generating the training data for our 2 frameworks - both regression tasks. EG3D synthesizes high-quality images of human faces from different views, through the ”tri-plane representation” of scenes. As this is, a method that leverages volume rendering and volume representation techniques, the 3D reconstruction of the face, is achieved by applying the marching cubes algorithm [**?**], an algorithm for creating triangular meshes from implicit functions. EG3D can also generate accurate depthmaps - d_{syn} of the synthesized human faces, providing the main ground truth data for training the depth regressor (see section 3.1)

EG3D is also used as the main rendering pipeline, the main backbone for 3D face mesh generation. By predicting the EG3D intermediate latent code w_{pred} (after the mapping network) of any given single-view human face image, and feeding it in the generator, a high-quality 3D shape of the respective input face image, is generated. This is visualized, also in Figure 3.6 above. EG3D given camera parameters - cam and and intermediate latent code - w, it generates a depthmap d_{syn} and the corresponding image I_{syn} with height and width of 128 pixels. The motivation behind regressing directly w, after the mapping network, instead of the random latent vector z, is that an intermediate latent code could be 1) fed directly to the generator without the use of a mapping network 2) an intermediate latent code represents more accurately the distribution of face data than the Gaussian space

3.2.4 PTI - inversion

EG3D can generate random face images, along with a 3D reconstruction of these faces. Nevertheless, external techniques are required, to ”feed” as an input a target face image, to leverage the representational and editing ability of EG3D. These methods, aim to solve the problem of projecting an input image, to a general GAN’s latent domain. GANs generate images, by sampling a random latent vector from their latent domain. This latent vector is parsed inside the generator and an image is synthesized. This is straightforward and a variety of synthetic images are produced. However, most GANs offer unique styling and editting capabilities, and users are interested in latent code inversion methods. Projecting an image to the latent space of a generator, allows a

custom image to be reproduced by the GAN, by leveraging simultaneously its editing and styling capabilities on this image. Most inversion techniques are based on minimizing an lpips or pixel-wise loss between the input image and the GAN generated image, by predicting the latent code. This works in some cases, however additional training of the generator can lead to more robust results. PTI [41] stands for Pivotal Tuning Inversion and it implements this simple idea. PTI works on a similar principle like most inversion methods. The image is projected to the latent space of the GAN and the respective latent code that can reproduce the desired image, is retrieved. This is achieved by optimizing the latent code via an LPIPS reconstruction loss and an l1 noise regularization. The novelty of "PTI", is that it also further fine-tunes the generator weights, based on this latent code, to improve reconstruction results even more. Simply stated, the initial latent code from produces a reconstruction that closely resembles the custom/target image. However there is still a significant distortion between the 2 images, and this distortion is termed as "identity gap". To bridge this gap, the generator is fine-tuned using LPIPS and L2 losses between the custom and generated image (based on the pivot latent code).

During fine-tuning of the generator, it is of prime important that the latent space retains its semantic properties required for editing and stylizing inverted image. As a result, locality regulation is introduced, which is a special regularization technique, to alleviate this issue. For example, during EG3D finetuning, the pivot is interpolated with a random style code sampled from the mapping network. The interpolated code, is fed to the finetuned EG3D and the original EG3D, generating respective images. Then by leveraging LPIPS and L2 losses between the 2 images, the generator learns to localize the changes to the "latent space only around the pivot" [41]. This way some form of generator regularization, ensures that the rest of the latent space remains unchanged and EG3D stylizing and editing capabilities are preserved. This method, does not allow for common baseline comparison of different GANs, however PTI is implemented in EG3D to reproduce custom images. By reproducing a custom/target image EG3D can generate also, the depthmap of this image. This is quite useful for evaluation purposes, especially for the depthmap regression model. For the evaluation of the latent code regression model, the pivot latent code can be used as ground truth (without generator finetuning), as the generated images (from this pivot) closely resemble the target/custom image, without further generator retraining.

PTI-inversion, involves retraining the generator, as well as projecting the image to the EG3D latent space via latent code optimization, resulting to slow execution times. PTI is very accurate and robust, however it is quite slow, as the two-step inversion takes around "3 minutes on a single Nvidia GeForce RTX 2080" [41]. This is the motivation behind the latent code regression model, to develop a framework to obtain a pivot latent code, with no further delay. This may enable face shape reconstruction for video data, by parsing the (predicted) inverted latent code in the EG3D for each frame, or for real-time 3D face reconstruction.

3.2.5 UNET

The framework is implemented by leveraging the UNET architecture, that is used mainly in image segmentation tasks. UNET has a U shape architecture, consisting of 4 encoder and 4 decoder networks, that extract high-frequency as well as low frequency details. This means, that it can capture features that represent the overall shape of the face, as well as the facial details. Overall UNET is a U-shaped encoder-decoder network architecture with:

- 4 encoder blocks (contracting path)
- 4 decoder blocks (expansive path)
- a bridge connecting the encoder-decoder blocks

An overview of the UNET architecture is shown in Figure 3.9. The U-shape architecture consists of a contracting path and an expansive path. The contracting path, consists of a series of encoding blocks that extract feature maps, by halving their spatial dimensions, at each step, and doubling the number of feature channels (filters). Each encoder block, extracts features from the input image, by passing the input image

through 2 convolutional layers of kernel size 3x3, followed by a ReLU activation function. The ReLU output, acts as a skip connection for the respective decoder block, whereas max pooling of size 2x2, is performed and the output is fed to the next encoder block. The use of ReLU activation function, in general introduces non linearity into the network and results to lower generalization error, whereas the max poolling layer, is used to aggregate the feature map and reduce the number of trainable parameters.

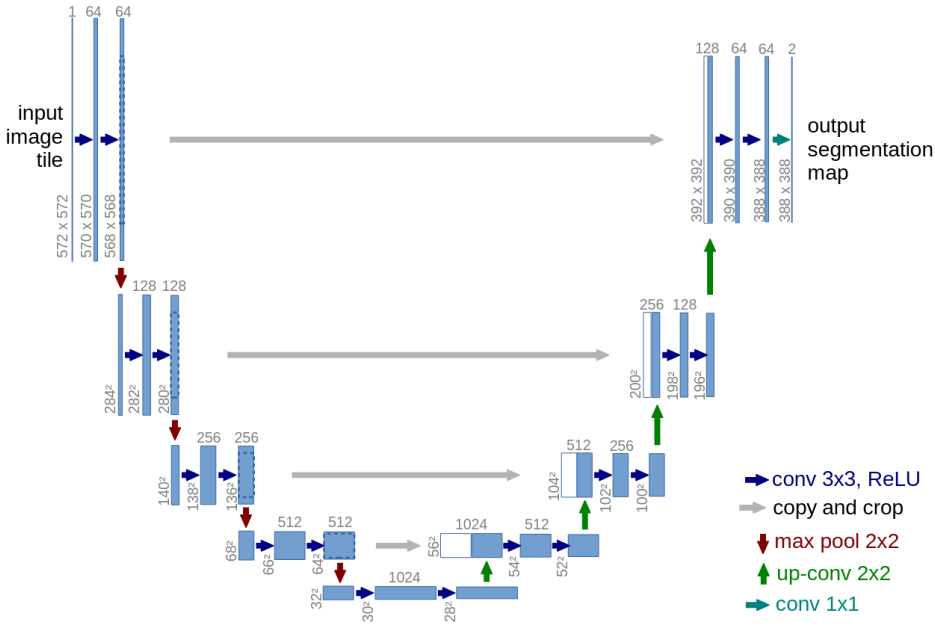


Figure 3.9: UNET architecture. It consists of a contracting path (left side) and an expansive path (right side). The contracting path, follows a series of feature map downsampling, with double number of filters at each downsampling step. The expansive path, involves a series of upsampling steps, with additional feature information form the skip connections. Each upsampling step, invovles an "up-convolution", with the number of spatial feature map dimensions doubling. The blue boxes are the feature maps, whereas the number of filters (channels) of each feature map is shown above the box. The feature maps spatial dimension is indicated next to each blue box. The cropping of the feature maps, from the skip connections, accounts for the boarded pixel loss due to the convolution operations. Image taken from [42].

The trick of including skip connections, results to more detailed semantic features being generated by the decoder. Skip connections, provide additional feature information to the decoder network, such as low-level features e.g edges, corners, that can help the decoder achieve a more precise localization. They act as a shortcut connection, that allows the indirect flow of gradients to earlier layers during backpropagation and hence results to better overall representation. The encoder-decoder bridge, allows the flow of information from the contracting path to the expansive path and the respective encoder, to encoder blocks. The bridge consist of 2 convolutions with kernel size 2x2, followed by a ReLU activation. The feature maps, are passed through a series of 4 decoder networks, as part of the expansive path, during which feature map upsampling is achieved, by doubling the feature map spatial dimension and halving the number of feature map channels(number of filters). Each decoder block, inputs feature maps, that correspond to an abstract representation of the face, and generates a feature map that corresponds to more detail face representation with improved semantic details.

Specifically, each decoder block, consists of a 2x2 transpose convolution ("up-convolution"), followed by a concatenation of this output feature map, with the provided feature map from the skip connection. The concatenated feature map is fed to 2 additional 3x3 convolution layers, followed again by a ReLU activation function. The additional low-level features from earlier layers, provided by the skip connections, result to feature maps that account for facial details such as the mouth region, or noise region, wrinkles etc. This, allows

to generate latent codes, from features that can capture both the overall shape of the face, as well as facial details.

In the last decoder block, after the 2 ”up-convolutions”, the feature map is passed through a 1x1 convolution, to generate feature vectors. In the original UNET formulation, the output from the last decoder block, is passed through sigmoid activation function to generate a segmentation mask. Due to the reason, that UNET was initially used, for segmentation tasks, and the original formulation involved a sigmoid output of the class probabilities, related to the task.

For regression EG3D latent code, the UNET original architecture is used, with some minor modifications, to account for the model output. The output is a latent code (EG3D latent code) of size (1, 512, 14). The main modification involve:

- Removing the last 3 decoding blocks from the expansive path
- Adding a series of extra convolutions of varying size to bring to desired output shape
- Add extra max pooling layer to bring to desired output shape

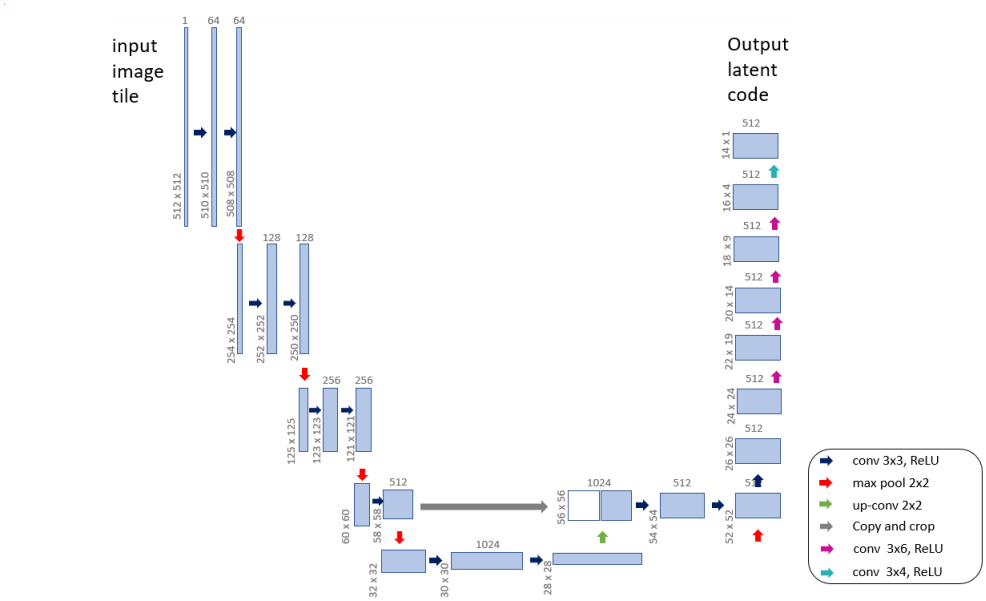


Figure 3.10: UNET modified architecture, for latent code regression. The model follows the same logic as UNET, but to match the required output size format, the 3 decoding blocks are replaced with additional convolutional and max pooling layers.

Removing the decoding blocks, does not allow the model to create feature maps, by leveraging low-level features from the earlier layers, through the skip connections. However, by including this extra decoding blocks, it would not be possible to concatenate the skip connection feature maps, to the expansive path feature maps, and retain the number of filters to 512, required for the output size. This is due, the mismatch of the skip connection feature maps, and expansive path feature maps, because of the pixel boundaries being lost throughout the convolution process. Simply stated, that given an input of (512, 512, 3), following the standard UNET model, will result to an output of (h, w, c), with h and w, smaller than 512, and hence the model won't be able to account for the output size of the latent code (1, 512, 14), in at least 1 dimension. The extra convolutional layers, are added to progressively, bring the model output, to the desired output size. The output can be seen, either as a feature map with spatial dimension of 512x14 and with 1 feature channel, or it can be see as a feature

map of 1x14 spatial dimension with 512 filters/feature channels. The latter is the most appealing option, as the output EG3D latent code, does not require dense predictions, rather than a more general solution. The output of size (1, 512, 14) is predicted by the model as a feature map with 512 filters of spatial dimension 1x14.An overview of the modified UNET for latent code regression, can be shown below (see Figure 3.10).

3.2.6 ViT architecture

This model architecture, leverages the novel Vision Transformers (ViT), developed by Google [54]. The Vision Transformer works in the same principle wih a traditional transformer for NLP applications [53]. This model, inspired a change in the field of vision related tasks, as it aspired to replace the traditional CNNs, by presenting vision transformers as a a better alternative for image classification. Transformers can be thought as a generalization of MLP, (whereas CNNs as explicit specializations of it), that can generalize very well, due to strong inductive bias. Transformers are models that operate on sequences, on sets, that we call tokens. The transformer inputs these tokens and performs an attention operation. In the case of feeding in directly - as tokens, all the pixels of the image (”raster of pixels”)- transformers would be limited by the memory and compute requirements of the quadratic attention (inner product between all tokens). As a result, for the vision transformer, instead of requiring any single pixel to attend to every pixel in the image, we input tokens that correspond to non-overlapping patches of image. The difference of Vision Transformers, with CNNs, is that instead of doing local attention across all pixels, they perform global attention, by leveraging image patches. The main architecture of a Vision Transformer is shown in Figure 3.11.

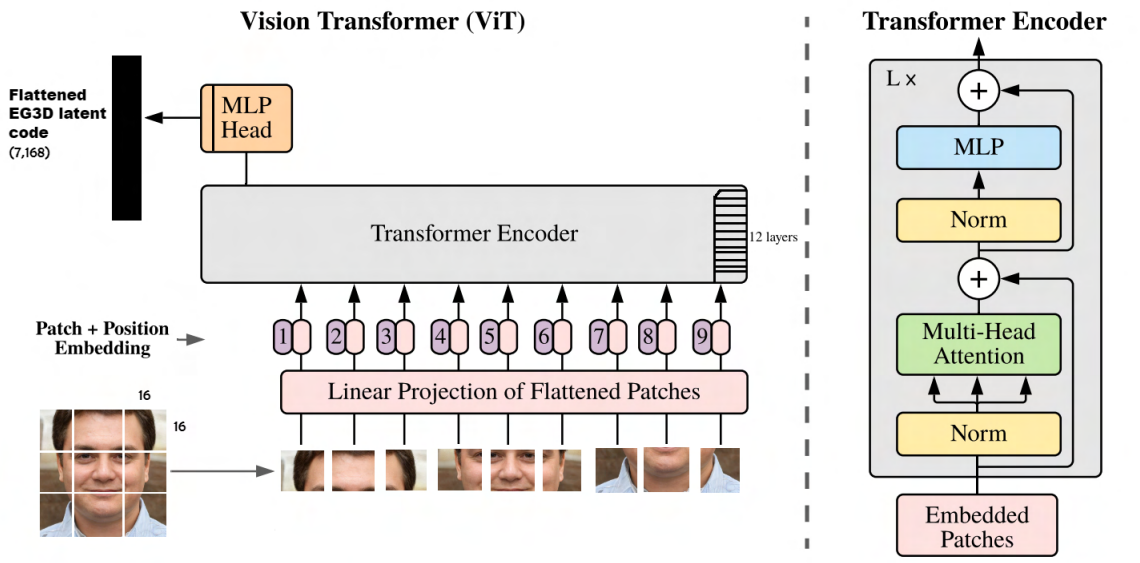


Figure 3.11: ViT architecture for EG3D latent code regression (right). An input image is divided to patches, that are flattened and linearly projected, to produce the path embedding (commonly known as tokens). The patch embeddings are augmented with position embeddings. These tokens, are then fed to the transformer encoder (detailed visualization) and an Attention module, mainly captures more global features, by going through image patches. Transformer encoder consists of 11 layers of similar grey blocks (shown left).The transformer output is fed to an MLP head (consisting of a normalization and fully connected layer), that recovers the a flattened version of w - EG3D latent code. Image adapted and inspired by [54].

The input face image, is divided to a number of non-overlapping patches (16x16). These patches are unrolled into a sequence. Each patch is vectorized and linearly projected, to produce the patch embeddings.

These patch embeddings, in the transformer literature are called tokens. These tokens are augmented with some position embeddings. Position embeddings are used to parameterize the image patches, as the transformer is parametrization invariant, due the model weights being calculated on the fly, during the multi-head attention. Positional embeddings, enumerate the sequence patches (1D positional embeddings), and hence the entire vectorized patch is retrieved. Both patch and position embeddings-tokens are fed as input to the transformer encoder. The transformer encoder consists of many transformer blocks/layers (12 layers), similar to 3.11 (right). The output of the transformer encoder, is fed to an MLP Head, that consists of a normalization layer and a fully-connected layer. The final model output is a flattened style vector of size (7168), which can be converted to the original EG3D latent code of size (512, 14). Specifically, ViT Base architecture (ViTB16) is leveraged for the latent code regression task [2] [46], which consists of 12 transformer layers, and a patch embedding dimension of 1024. Overall, this model architecture is used in the main latent code regression method, due to the reason that Vision Transformers perform comparably (if not better) to CNNs.

Chapter 4

Experiments and Results

The main experiments and their results are reported in this section. The main 2 frameworks aim to solve a depth regression task and a latent code regression task, in order to achieve 3D reconstruction from a single-view face RGB image. Other models, have achieved similar results, with leveraging different techniques. In this section the focus is on presenting the various experiments to obtain a robust depth regression model (see section 4.3), as well as reporting training, validation and test results. Model performance is assessed on both synthetic and real "in-the-wild" faces images, for both tasks. Additionally, the relevant experiments for the EG3D latent code regression task, are described below (see section 4.4), along with the respective reported results. For both tasks, hyperparameter tuning is performed, as well as different neural architectures described in the Method section (Chapter 3) are investigated, with varying training dataset sizes. More detailed experiments that are not directly related to the training or evaluation of the depth regression and latent code regression models, are presented in Appendix A.4,A.5, A.6, A.7. Nevertheless, these are experiments that helped build intuition around the problem, offered inspiration to tackle the problem and implemented techniques from other methods, to achieve a full 3D reconstruction pipeline. The interested reader is referred to the Appendix section for more details.

4.1 Prosopo

To assess the performance of the depth regression and latent code regression frameworks, a common evaluation dataset is employed. This test dataset, consists of the input and ground truth data required for both tasks. For the depth regression task, to successfully evaluate the framework performance for both Attention - UNET and the MiDaS-DPT neural network architectures, we need:

- an input RGB image I_i of a human face
- a depthmap d_i corresponding to the input image I_i

As for the latent code regression task, both models UNET and ViT are assessed on the same evaluation dataset, that should consist of:

- an input RGB image I_i of a human face
- an EG3D latent code w_i corresponding to the input image I_i

Prosopo is our main evaluation method between our different models, and is an objective way to assess the various experiments of our various models. Prosopo (means "face" in Greek) and consists of 1600 high-quality "in-the-wild" real RGB face images , manually selected from the FFHQ dataset [24]. There has been a meticulous selection and filtering of the initial FFHQ dataset based on criteria described below, in an attempt to form a dataset to fairly evaluate performance. All our models, for both tasks, have been trained on mainly front-view (anfas) images, of tightly cropped faces. Therefore, the filtering process involves removing any

Figure 4.1: Visual examples from the ”prosopo” dataset used for model evaluation.

images that have extreme side-view faces (profile view $\pm 90^{\circ}$ rotation). Another filtering criterion is, that the human face is not obstructed by any objects or people (e.g microphone, hand gestures, other humans). The ground truth depth maps are generated from EG3D, which is leveraging volumetric neural represenation and volumetric rendering, to recover the entire face shape, ignoring irrelevant occlusions. Nevertheless, the depth regression model, just predicts the depth directly for every pixel and hence is sensitive to occlusions (mismatch between ground truth depthmap and predicted depthmap).The dataset, still consists of some challenging images, to evaluate the performance under ”in-the-wild” conditions, but the main purpose of ”prosopo” is to evaluate face shape reconstruction in relatively ”ideal conditions” (good light conditions, frontal-view faces). The model, is not trained on occluded face images and it is not expected to perform well in an unknown domain like this. Nevertheless, the dataset consists of human faces with accessories, such as hats, glasses, or hijabs and the model performance is evaluated on inferring its 3D structure. Examples from the prosopo dataset can be shown in the Figure 4.1. Finally, the ”prosopo” dataset consists of a limited number of very challenging images, such as hand occlusions or occlusions by other humans, in order to further assess the robustness of the latent code regression task, in disentangling the main face from the rest of the occlusions or the image background.

The main difficulty in creating this dataset, was not the RGB image selection or filtering. The most challenging part was to obtain the respective depthmaps - $d := \{d_1, ..., d_{(i=N)}\}$ and latent codes $w := \{w_1, ..., w_{(i=N)}\}$ for all N images. Inversion GAN techniques such as PTI-inversion are leveraged, to project the input image to EG3D's domain. As a first step of PTI, the pivot latent code for every input image is obtained. This pivot latent code is stored and used as our w ground truth latent code for our evaluation dataset. To recover the corresponding depthmaps d for our input images $I := \{I_1, ..., I_{(i=N)}\}$, the second step of the PTI-inversion method is carried out. Even though the pivot latent code, reconstructs the input image very robustly, to use the corresponding depthmaps as ground truth, a perfect reconstruction is required. Hence, the EG3D generator is further-finetuned, until the input pivot latent code and the respective camera pose pose parameters, yield a perfect image reconstruction. After a perfect match, between the input images and the synthesized images is obtained, the respective generated depthmap, can be used as the dataset's ground truth. The assumption, for creating this dataset, is that EG3D produces robust 3D representations of human face images, along with their respective depthmaps. The pivot latent code, does not yield perfect image reconstruction in all cases, however are results are almost perfect (see A.2). For more details about the PTI-inversion process and the ”prosopo” dataset, please refer to A.1. Overall, the ”prosopo” dataset is one of the key contributions of this paper, with input RGB face images, matched with high-quality depthmaps and the respective EG3D latent code.

4.2 Training & Validation Data

The models are trained and validated on synthetic face data, generated by EG3D. EG3D is pretrained on the FFHQ dataset and can generate high-quality face images. The genererated data, are mostly RGB images of size (512, 512, 3), with height and width of 512 pixels and 3 color channels. The generated faces are mainly tightly-cropped faces, as the emphasis is given on learning the face region details (e.g eyes, wrinkles, mouth). However, to improve the robustness of the model, there is a significant amount of face images with a variety of different camera poses and face angles, included both in the training and validation dataset. As one of the 2 tasks focuses on estimating the depth of a human face, emphasis is given to training on frontal-view human faces. This enables also the model to reconstruct most of the human face, based on a depthmap. Frontal-view face images (or else called Anfas), hold most of the face information to recover the entire human face image. In case of profile-view face images, the predicted depthmaps, infer 3D structure of only one side of the human face. As a result, during our training, we included a limited number of side-view face images, as this could further confuse the model. It needs to be underlined, that extreme face side-views (90° rotation of face), are not very well modelled by EG3D itself. This can result to low-quality data (see Appendix A.3) that according to George Fuechsel (IBM programmer) and the principle of "GIGO" (Garbage In Garbage Out), will lead to poor model training, generalization and performance. A detailed breakdown of the various face angles and camera poses, as well as their respective percentage of the entire dataset, is shown in Table 4.1.

Category	% of Training Dataset	% of Validation Dataset
Frontal view - zero pose(0° rotation) - tightly cropped	0.7	0.8
Side view ($> \lvert 70^\circ \rvert$ rotation) - tightly cropped	0.06	0.05
Side view (between $\pm 70^\circ$ rotation) - tightly cropped	0.12	0.05
Frontal view - zero pose(0° rotation) - neck and face view	0.12	0.1

Table 4.1: The table describes the portion of data from the various categories, for the training (N=50000 images) and validation sets (N=1600 images). The visualization of each data category used in the training and validation sets, can be further visualized in Figure 4.2

Visual results for each category are depicted in Figure 4.2. As it is observed, frontal-view data, are high-quality synthetic images of a zero face pose (4.2 a). For extreme side views, the synthetic results are more noisy, especially around the neck and ear region (4.2b). For side views that are between $\pm 70^\circ$, the synthetic images have some minor distortion, but the general face shape and details are represented very robustly (4.2c). As for a different camera pose, that shows the entire head and neck, some noise in the hair region is observed (4.2d). Overall, the models are trained and validated with high-quality face image data, with a variety of camera poses, to improve the robustness of the models.

4.3 Depth Regression

Given a single RGB image of a human face, the geometry of the depicted face can be reconstructed. A model architecture is deployed to map the input image, to a depth value for each pixel of the image. Later, the depth map is leveraged to extract the 3D mesh of the face. This underlines the importance of the depth regression task. Predicting correctly depth values for an input RGB image, results to reconstructing robustly the 3D shape of the human face. Various experiments are conducted to investigate the best approach to the task of face depth regression.

(a) (b) (c) (d)

Figure 4.2: The various categories of the training and validation dataset, are visualized through the depicted examples. a) Frontal view - zero pose(0° rotation) - tightly cropped b) Side view ($> |70°|$ rotation) - tightly cropped c) Side view (between $\pm 70°$ rotation) - tightly cropped d) Frontal view - zero pose (0° rotation) - neck and face view

4.3.1 Experimental Setup

For the depth regression task, the main 2 model architectures, that we experimented with, are the MiDaS-DPT and UNET-Attention neural network architectures (see sections 3.1.4 & 3.1.3). Both models are trained in a supervised manner, with the RGB face image inputs and their respective depthmaps. The training data for the depth regression task are: a) synthetic EG3D RGB face images of size (512, 512, 3) which are resized to (128,128,3) b) synthetic depthmaps of size (128,128). Each depthmap stores relative depth values as normalized values in the range of [-1, 1]. This tackles issues, with local optima (not reaching global optimum) and generally leads to faster convergence. The input image is also normalized with a mean of 0 and standard deviation of $\pm$ 1, scaled between the range [-1, 1]. For more details refer to section 3.1.2. The models are trained with an Adam optimizer fine-tuned with learning rates of $lr = [0.1, 0.01, 0.001, 0.0001]$ and momentum values - $m = [0.8, 0.9]$. The models are trained for various epochs - $e = [0, 5, 10, ..., 100]$ for batch sizes - b of 1 and 5.

The main experiments for both models, involve an overfitting task and a purely training task. The aim of overfitting the model, is to investigate the lower boundary for the generalization error of the model. Over-fitting the models, assesses the feasibility of achieving good training and evaluation results. Inability to overfit a model to a small number of samples, proves that with the given data or model architecture (not enough model complexity or capacity), good training results on larger sample sizes, cannot be achieved. Overfitting the 2 models, is implemented for training sample sizes of 1, 10 and 100. The results are reported below.

As for training the 2 model architectures, with higher number of training samples, such as 1000, 10000, 20000 and 50000 samples, the training, validation and evaluation results are presented below. Overall, for both model architectures, for the depth regression task, the regressor optimizes an l1 per pixel loss, between the synthetic GT depthmap (produced by EG3D) and the predicted depthmap (for the same input image), both of size (128, 128). The model is optimized, by minimizing a pixel l1-loss on a synthetic training set. Specifically the l1-loss is applied to all output and ground truth depth maps and is given by the mathematical formula :

$$l1_{loss} = |y - \hat{y}|_1 \tag{4.1}$$

$$l1_{loss} = \sum_{i=1}^{w} \sum_{j=1}^{h} y_{ij} - \hat{y}_{ij} \tag{4.2}$$

where $\hat{y}$ refers to the predicted output depth map, y refers to the ground truth depth map and h,w to depthmap height and width respectively. The l1-loss is given by the absolute value of the difference between the predicted and true depth value for each pixel.

For the depth regression task, the reported results for training, validation and evaluation/testing, are based on the same l1 per pixel loss of predicted and ground-truth depthmaps. To further test, the performance of the model on the evaluation set, the input "in-the-wild" face images from the "prosopo" dataset, are further preprocessed. This entails, implementing a background removal method such as RVM [28] or through an online tool such as [3]. In the mean l1 score calculation, only the masked region (defined by background of the image) of the predicted and ground truth depth image, is taken into account. This further investigates the high-quality reconstruction ability of the models on real "in-the-wild" data, in some more ideal "conditions".

4.3.2 Results

The results on training the depth regressor are presented in this section, for both model architectures (MiDaS-DPT and Attention-UNET). The validation set consists of 1600 samples, whereas the evaluation set is the "prosopo" dataset explained above, consisting, as well of 1600 samples.

Quantitative

For both the MiDaS architecture and Attention-UNET, extended hyperparameter tuning is implemented across different learning rates ($lr = 0.1, 0.01, 0.001, 0.0001$) and momentum values ($m = 0.9, 0.8$). All experiments demonstrated that the best hyper-parameters have a learning rate of $lr = 0.001$ with $m = 0.9$. Both architectures yield better results for a batch size of 5. For these fixed hyper-parameters we present, the training loss along with the validation and test scores, for each model. Each model is trained on different number of samples (N_{train} = 1, 10, 100, 1000, 10000, 20000, 50000). For each experiment, we report the best validation score, with its corresponding training score for a specific number of epochs (varies across different experiments with different training samples). For each architecture, the best model (optimal weights) corresponding to the best validation, is evaluated further by reporting the test score on the prosopo dataset.Each score is the average l1-per pixel loss across all set samples. The results for both architectures are shown in Table 4.2 below.

		MiDaS-DPT			**Attention-UNET**		
e	N_{train}	train ↓	val ↓	test ↓	train ↓	val ↓	test ↓
3000	1	0.022	0.222	0.276	0.034	0.375	0.48
1000	10	0.025	0.198	0.026	0.041	0.108	0.225
1000	100	0.029	0.067	0.206	0.078	0.088	0.221
200	1000	0.074	0.085	0.204	0.069	0.081	0.212
20	20000	0.064	0.062	0.205	0.022	0.068	0.217
20	50000	0.032	0.054	0.187	0.035	0.038	0.169

Table 4.2: The table summarizes the validation, testing and corresponding training scores for the 2 model architectures. Each model is trained for a number of epochs - e for a training sample size of N_{train}. The training and validation mean l1-losses are reported for both architectures (the lower, the better). The validation loss is reported for a validation set of 1600 synthetic images. The testing loss is the mean l1 per pixel loss reported on the prosopo dataset. For N_{train} of 1, 10 and 100, the model is strictly overfitting to these samples and the feasibility of the task is assessed. For the remaining experiments, the number of epochs are reduced and adjusted to avoid the extreme overfitting phase.

According to the table results, both models achieve perfect results for the overfitting tasks. The training losses for a training size of 1 and 10 are below a mean l1-loss of 0.041 (for both models) and perfect depth reconstruction. This indicates that both models have the capacity to achieve great results on larger training sets with good generalization. MiDaS-DPT achieves a low validation score of mean l1-loss of 0.054, whereas Attention-UNET, scores a lower validation mean l1-loss of 0.038 . Both models, perform better for the last

"row" of the table, where the training size is larger. The validation score is reported, on a validation set that consists of EG3D synthetic images, similar to the training data (for more details refer to section 4.1). On the other hand, the test score, is the reported score, of each model on the evaluation dataset "prosopo" (for more details refer to section 4.1). MiDaS-DPT achieves an mean l1 score of 0.187, whereas the Attention-UNET improves on this score by reaching a mean l1 score of 0.169 . As expected, the best scores are achieved for the model trained with higher number of data, on account that supervised methods are mostly data-driven methods.

The evaluation score improves for both models for 50000 samples training size, however this is not only attributed to the quantity of the data, but also to the quality of the data. Simply stated, for the first 20000 samples, only zero-pose frontal faces, were included in the training set. In the 50000 samples of the last training, a variety of data, are added, with different camera poses, that allow the models to generalize better to "in-the-wild" images. The best models are summarized for both architectures and the evaluation score is reported in Table 4.3 below. Results are presented for each model, when an extra preprocessing step-of removing the image background-is implemented to the input RGB images. This gives a better insight to which model, can disentangle more easily features related to the face foreground and the background of the image. Finally, removing the image background is implemented only to "in-the-wild" faces images present in the testset, as it includes images with more challenging backgrounds, that sometimes blend with the face foreground. According to the reported results, removing the background [28], does improve performance on the evaluation set for our models.

Models	test ↓
MiDaS-DPT	0.187
Attention-UNET	0.169
MiDaS-DPT + RVM	0.183
Attention-UNET + RVM	0.162

Table 4.3: The table summarizes the evaluation mean l1 scores (test) for the best MiDaS and Attention-UNET models. Evaluation results are also reported, for the same test set, with an extra preprocessing step of removing the image background. Each evaluation score is the mean l1-loss between predicted and ground-truth depthmaps corresponding to the input images of the test set (with background removal - RVM [28]). Evaluation results for applying background removal on the input images, improves the performance of both models (lower is better).

The mean l1 loss for both models, is not calculated for the entire depth image, but only for the binary-masked region - defined by foreground and background. Both models, perform better with RVM, however the performance of the Attention-UNET (mean l1 loss of 0.162) is far more superior than the performance of MiDaS-DPT (mean l1 loss of 0.183), as validated by the visualization of the results.

Qualitative

For deeper understanding of the quantitative results, let's delve into the visual interpretation of those. Given an input RGB face image, the models, predict the relative depth values for each pixel of this image. In Figure 4.3 and 4.4, example input images from the validation and test datasets, along with the corresponding predicted and ground truth depthmaps are visualized for the best MiDaS model. Along with the depthmaps, we present the corresponding reconstructed meshes for better visualization of the results. According to the figures, on the validation set, MiDS fails to model facial details completely (row 7). The glasses are not modelled at all in some cases (row 2, 6), whereas the ear region is not reconstructed at all (e.g row 8 & 3). Similar performance is observed on the testset visual results, with the eye and mouth region, completely being omitted by the model (row 8 & 5). General structure and hair silhouette is recovered by the model, which remains robust even with image of more complex backgrounds (e.g row 3 & 4).

Visualisation results of the best Attention-UNET model, on the validation and test sets are depicted in Figure 4.5 and 4.6. Attention-UNET emphasizes more on recovering facial details rather than the overall face structure, as seen from the visual results. This is a desired characteristic for our depth regressor, as the main focus is to reconstruct facial details that are person-specific instead of the general geometry of a human face. Attention-UNET performs extremely well on examples from the validation set, as close-to-perfect reconstruction is achieved (e.g row 2). The eye region for most examples is reconstructed (row 3, 4, 7), however results are not perfect.Hair geometry is recovered robustly, with desired details, even for long more complex hair (e.g row 4). Mouth and nose regions are also accurately reconstructed as seen in Figure 4.5 (e.g row 3). On the evaluation set (see Figure 4.6), the model, predicts robustly the relative depth for the face region and reconstructs accurately the facial details. General face and hair geometry is accurately captured by the model (e.g row 4, 5), even recovering the ear region (row 3). Results are comparable to the validation set visual results, nevertheless, there are some failure case around the eye region (e.g row 8).

Visual results prove that even though the validation and test scores are very close for both models, the quality of the depthmap and reconstruction results is higher for the Attention-UNET model. In general, the MiDAS architecture achieves low quantitative scores, however the model learns features that a related more to the general structure of the face. This refers, to the geometry of the facial attributes and the overall face silhouette. On the other hand, the Attention-UNET architecture predicts higher frequency details and learns features that can model facial details (such as the mouth region). A direct comparison of the qualitative results for both models is visualized in Figure 4.7.

Attention-UNET is by far the best performing model, as it can predict robust depthmaps, even for data of unseen poses. It is easier for the model, to accurately predict depthmap, in image instances with no background and foreground blending, that could hinder the model performance. The visual results in Figure 4.7, demonstrate some failure cases, where the model cannot model complex hair robustly (e.g row 5), for both models. However, Attention-UNET achieves some good reconstruction with regard to facial and hair detail (e.g row 1). Overall in row 2,3 and 6 there is a clear indication of the MiDaS-DPT architecture inferiority to the Attention-UNET architecture for the given depth regression task. In some cases, models cannot disentangle at all the background from the foreground, as it is seen in the hair region for row 3.

4.4 Latent code Regression

Given a single RGB image of a human face, the geometry of the depicted face can be reconstructed. A model architecture is deployed to map the input image, to an intermediate latent code in the EG3D domain. The latent code is later fed to the EG3D generator, along with camera pose parameters, to generate /reproduce the image appearance, implicitly render its 3D representation and reconstruct its geometry, including hair geometry.

4.4.1 Experimental Setup

For the latent code regression task , the main 2 model architectures, that we experimented with, are the ViT and the UNET neural network architectures (see sections 3.2.6 & 3.2.5). Both models are trained in a supervised manner, with the RGB face image inputs and the respective intermediate latent codes, as ground truth. The training data for the depth regression task are: a) synthetic EG3D RGB face images of size (512, 512, 3) or (384, 384, 3), depending on the model architecture b) the corresponding intermediate EG3D latent codes of size (14, 512). Each latent code is a an array of intermediate EG3D style vectors of size 512 elements. The input image is also normalized with a mean of 0 and standard deviation of $\pm$ 1, scaled between the range [-1, 1]. For more details refer to section 3.1.2. The models are trained with an Adam optimizer fine-tuned with learning rates of $lr = [0.1, 0.01, 0.001, 0.0001]$ and momentum terms - $m = [0.8, 0.9]$. The models are trained for various epochs - $e = [0, 5, 10, ..., 100]$ for batch sizes - b of 1 and 5.

The main experiments for both models, similarly to 4.3, involve an overfitting task and a purely training task. The aim of overfitting the model, is to investigate the lower boundary for the generalization error of

the model. Over-fitting the models, assesses the feasibility of achieving good training and evaluation results. Inability to overfit a model to a small number of samples, proves that with the given data or model architecture (not enough model complexity or capacity), good training result on larger sample sizes, cannot be achieved. Overfitting the 2 models, is implemented for training sample sizes of 100. The results are reported below.

As for training the 2 model architectures with higher number of training samples, such as 1000, 10000 and 50000 samples, the training, validation and evaluation results are presented below. Overall, for both model architectures, for the code regression task, the regressor optimizes an l1 loss, across all elements of the latent code block of (14, 512) size, for the predicted and ground truth latent codes. The gt code is extracted by generating a random face image with EG3D and storing its corresponding intermediate latent code. During, training of our regressor, the reverse process is implemented instead, and model training involves a face image as input and an intermediate EGD latent code, as output/label. The model is optimized, by minimizing this l1-loss on a purely synthetic training set of input face images. Specifically the l1-loss is applied to every element of the predicted and ground-truth codes :

$$l1_{loss} = |y - \hat{y}|_1 \tag{4.3}$$

$$l1_{loss} = \sum_{i=1}^{n}\sum_{j=1}^{s} y_{ij} - \hat{y}_{ij} \quad = \sum_{i=1}^{14}\sum_{j=1}^{512} y_{ij} - \hat{y}_{ij} \tag{4.4}$$

$$l1_{loss} = \sum_{k=1}^{v} y_k - \hat{y}_k \quad = \sum_{k=1}^{7168} y_k - \hat{y}_k \tag{4.5}$$

where $\hat{y}$ refers to the predicted latent code, y refers to the ground truth latent code and s, n to the style vector size (512) and the number of style vectors (14) respectively.The l1-loss is given by the absolute value of the difference between each element in the intermediate predicted and ground-truth latent codes of size (14, 512). However, it can be calculated as a per element absolute difference of each vector element for a flattened intermediate latent code of size v (14*512 = 7168 elements).

The reported results for training, validation and evaluation/testing, are based on the same l1 loss on predicted and ground-truth latent codes. In the training and validation set, there are input RGB synthetic face images. For the test set, the prosopo dataset is selected, therefore the models are evaluated in real "in-the-wild" images. In the case, that 2 latent codes, closely align, the respective generated images-from feeding the codes to the EG3D generator under the same camera parameters- will yield high resemblance and perceptual similarity. Similar generated image appearance, results to similar reconstruction results of face geometry. Hence, the model performance can be evaluated, by directly comparing the target/input image (fed in the regressor) and the reproduced image (generated from the corresponding predicted latent code fed as input to the EG3D generator). UNET is trained by minimizing the mean l1-loss over the EG3D intermediate latent code directly (14, 512), treating the code as 14 different style vectors with 512 features (see equation 4.4). On the other hand, ViT address the optimization problem, as directly estimating 7168 features from the input RGB image. This approach to the optimization problem, is derived from the formulation of the problem, as ViT predicts directly a flattened version of the intermediate EG3D latent code. This means that ViT predicts a single vector of size (7168, 1) (see equation 4.5) , which is then resized to (14, 512), in order to be leveraged by the EG3D generator at a later stage.

4.4.2 Results

The results on training the depth regressor are presented in this section, for both model architectures (ViT and UNET). The validation set consists of 1600 samples, whereas the evaluation set is the "prosopo" dataset explained above, consisting, as well of 1600 samples.

Quantitative

For the both architectures, extended hyperparameter tuning is implemented across different learning rates ($lr = 0.1, 0.01, 0.001, 0.0001$) and momentum values ($m = 0.9, 0.8$). All experiments demonstrated that the best hyper-parameters have a learning rate close to $lr = 0.001$ with $m = 0.9$. Both architectures yield better results for a batch size of 5. For these fixed hyper-parameters we present, the validation, training and test loss, for the models trained on different number of samples (N_{train} = 100, 1000, 10000, 50000). For each experiment, we report the scores for each model for a certain number of epochs (varies across different experiments with different training samples). Each score is the average l1-loss between the predicted and true latent codes, across all samples. The results for both architectures are shown in Table 4.4 below.

		ViT			**UNET**		
e	N_{train}	train ↓	val ↓	test ↓	train ↓	val ↓	test ↓
100	10	0.022	0.222	0.391	0.016	0.207	0.383
100	1000	0.025	0.198	0.310	0.003	0.217	0.299
20	10000	0.074	0.174	0.307	0.069	0.177	0.295
15	50000	0.045	0.131	0.281	0.041	0.155	0.273

Table 4.4: The table summarizes the training loss, along with the validation and testing scores for both model architectures. Each model is trained for a number of epochs - e for a training sample size of N_{train}. The training and validation mean l1-losses are reported for both architectures (the lower, the better). The validation loss is reported for a validation set of 1600 synthetic images. The testing scores are the mean l1-norm reported on the prosopo dataset. For N_{train} of 100, the model is strictly overfitting to these samples and the feasibility of the task is assessed. For the remaining experiments, the number of epochs are reduced and adjusted to avoid the extreme overfitting phase.

The reported results on the evaluation set, are the best scores from the entire 15 training epochs. The evaluation results do not correspond to the model that achieved the best validation score. The reason for deviating from the standard way of presenting the scores, is to quantitatively support that the models improve their evaluation performance, however after a number of training epochs, even though the validation scores keep decreasing, the models overfit to the training distribution and the testing scores, gradually increase. According to the table results, UNET achieves a low validation score of mean l1-loss of 0.155, whereas ViT, a mean l1-loss of 0.131 . Both models, perform better for the last "row" of the table, where the training size is larger. The validation score is reported, on a validation set that consists of EG3D synthetic images, similar to the training data (for more details refer to section 4.1). On the other hand, the test score, is the mean l1-loss, of each model on the evaluation dataset "prosopo" (for more details refer to section 4.1). For the evaluation dataset, UNET achieves an mean l1 score of 0.281, whereas ViT improves on this score by reaching a mean l1 score of 0.273 . The best scores for both the validation and test set, are observed for both models, for a higher training size. This is expected, as this is a data-driven task, with direct data supervision, but is attributed also to adding a variety of image faces. Similar to the depth regression task, in the first 20000 samples, only zero-pose frontal faces, are included in the training dataset. Including all 50000 samples in the training, leads to a variety of data, with different camera poses, that allow the models to generalize better to "in-the-wild" images. Adding a variety of data, aims to improve the performance of the models, to real "in-the-wild" images, which is indicated by the evaluation score on the prosopo dataset. However, both models, do not perform as good, as in the validation set.

For deeper understanding of the this performance observed in our best models, an analytical graph demonstrating the training, validation and test mean l1-losses with respect to the number of training epochs, for both models is presented in Figure 4.8.

As observed, both models, achieve a low validation loss, that steadily follow the training loss (or at least the trend). According to the plot, the test loss, drops at the first few training epochs, and in later epochs, the

test loss increases. This leads to a bad performance, in the latent code prediction task and does not yield robust visual results. The trend of decreasing validation loss, but increasing evaluation loss, indicates that the model overfits on the training distribution. Training and validation sets, consist of only synthetic images, whereas the testing set consists of only real images. As a result, the model needs to be trained on the distribution of real "in-the-wild" data, to improve its performance in the "prosopo" dataset. Fine-tuning of the existing best models on real data, is implemented for a small sample size of 1500 samples. Nevertheless, the improvements are not very significant (0.001 and 0.002 loss improvement for ViT and UNET on the evaluation set), hence larger training sizes of real data, are required to properly assess the effect of real data fine-tuning on the performance of our models.

Qualitative

To better understand the performance of the respective models, a closer inspection is required, to the visual representation of these reported scores. Given an input RGB face image, the models, predict the latent EG3D code, that is fed to the EG3D generator, and synthesizes the face image, obtains the volumetric representation of the face and can reconstruct the face geometry and appearance. The purpose of both models is, to reproduce the same image, given as input to the latent code regressor. Good latent code predictions are reflected on the visual similarity of the synthesized image and the input/target image. In Figure 4.9 and 4.10, example input images from the validation and test datasets, along with the generated images from the predicted respective latent codes are visualized for both model architectures. The validation set consists of synthetic images and the visual comparison between the input face image and the generated ones, can be viewed. The same applies for the test visualization results, for the generated images, for the target real face images from the prosopo dataset.

For the validation set, the reproduced images, have close resemblance in most examples. Details such as background color or some facial hair are not perfectly reproduced (e.g row 1 example #2, row 2 example #1 & #2, row 5 example #1). Both models fail to disentangle background and face foreground details, as shown in row 1 example #2 in Figure 4.9. Another failure cases is observed for examples of faces, with accessories such as hats (row 3 example #2). Nevertheless, both models perform robustly on the validation set, reflecting the quantitative good performance. Some failure cases are presented, to demonstrate any qualitative difference between the 2 models. For the validation set, there is some mere indication that ViT performs slightly better, which can be verified by the quantitative results.

Both models, perform poorly in the evaluation set, with unseen images, from a distribution totally unseen to our models. Even though the models recover basic semantic features and person expressions in most cases, a high identity resemblance in most examples, is lacking. According to the visual results, the skin color, the gender and the hair length is robustly represented in both the target and generated image (e.g row 3 & 4 examples #1 and #2). Furthermore, some examples, demonstrate that the model can recover, target-specific expression, the eye color as well as facial hair, such as beard (e.g row 2 example #1, row 6 example #1). Overall, it is clear that the models, can extract useful face features from the images and predict appropriate latent codes, however the evaluation performance cannot recover the entire person identity robustly. The visual test results, endorse the quantitative test results, as visually the ViT and UNET models perform comparably the same.

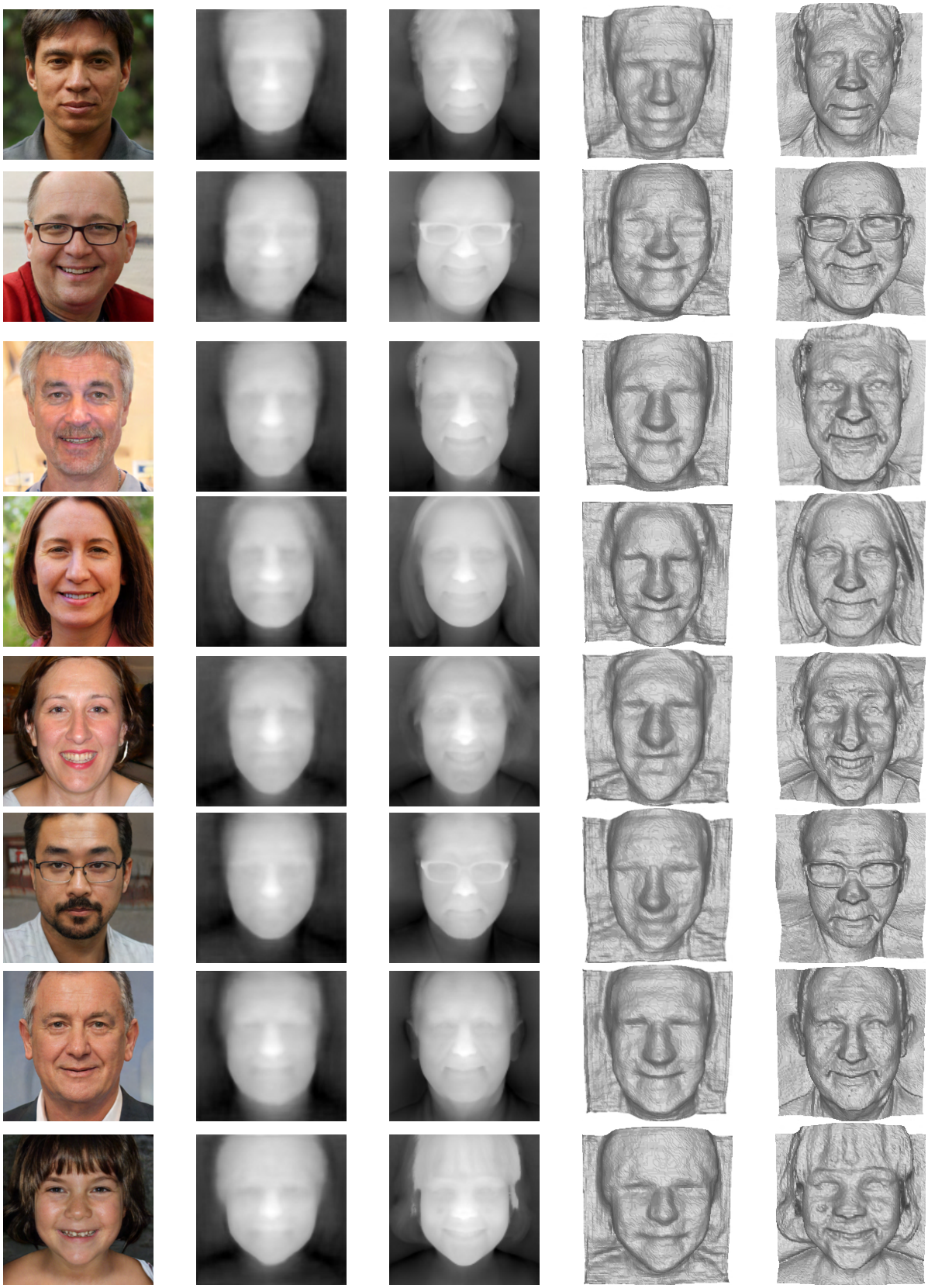
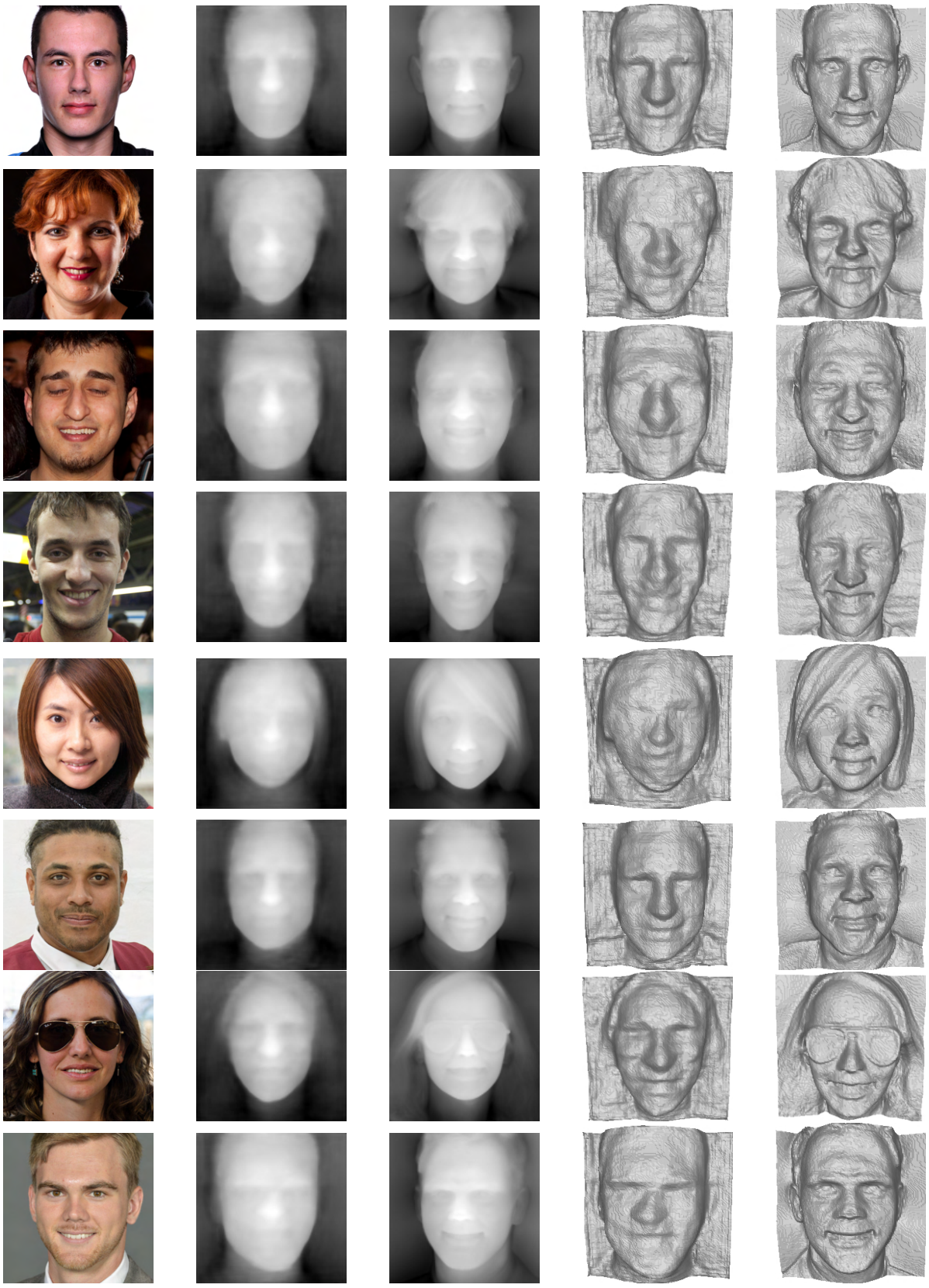


Figure 4.3: Visual validation results on synthetic EG3D data for the best MiDAS model. The input RGB image (a) is fed to the regressor, to output the corresponding depthmap (b). The ground-truth depthmap(c) is depicted, along with its 3D reconstruction (e). The predicted reconstructed mesh of the input face image is shown in d).



(a) (b) (c) (d) (e)

Figure 4.4: Visual evaluation results on real "in-the-wild" data from "prosopo" for the best MiDAs model. The input RGB image (a) is fed to the regressor, to output the corresponding depthmap (b). The ground truth depthmap (c) is depicted, along with its 3D reconstruction (e). The final predicted reconstructed mesh of the input face image is shown in d)

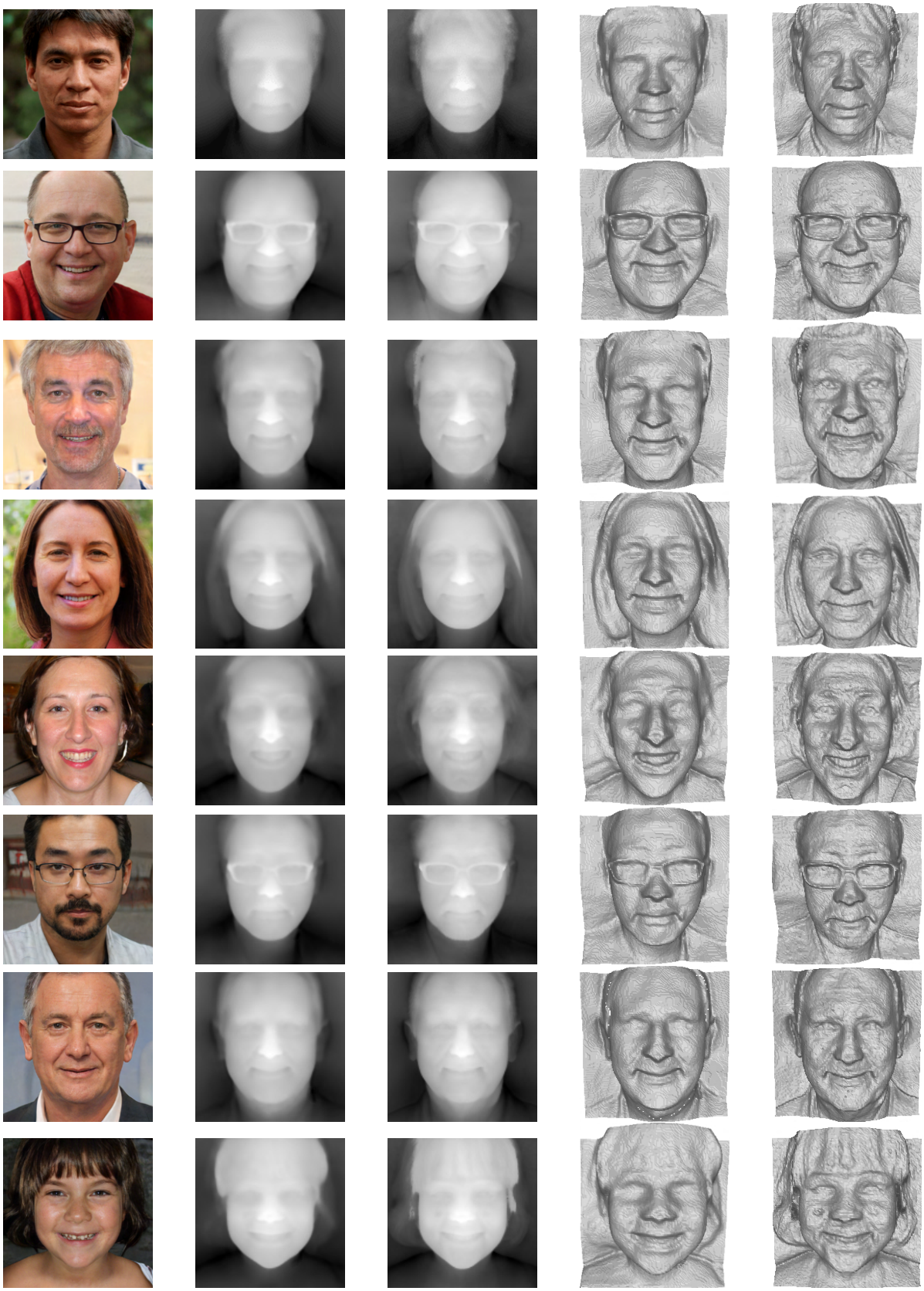


Figure 4.5: Visual validation results on synthetic EG3D data for the best UNET model. The input RGB image (a) is fed to the regressor, to output the corresponding depthmap (b). The ground truth depthmap (c) is depicted, along with its 3D reconstruction (e). The predicted reconstructed face mesh is shown in d).

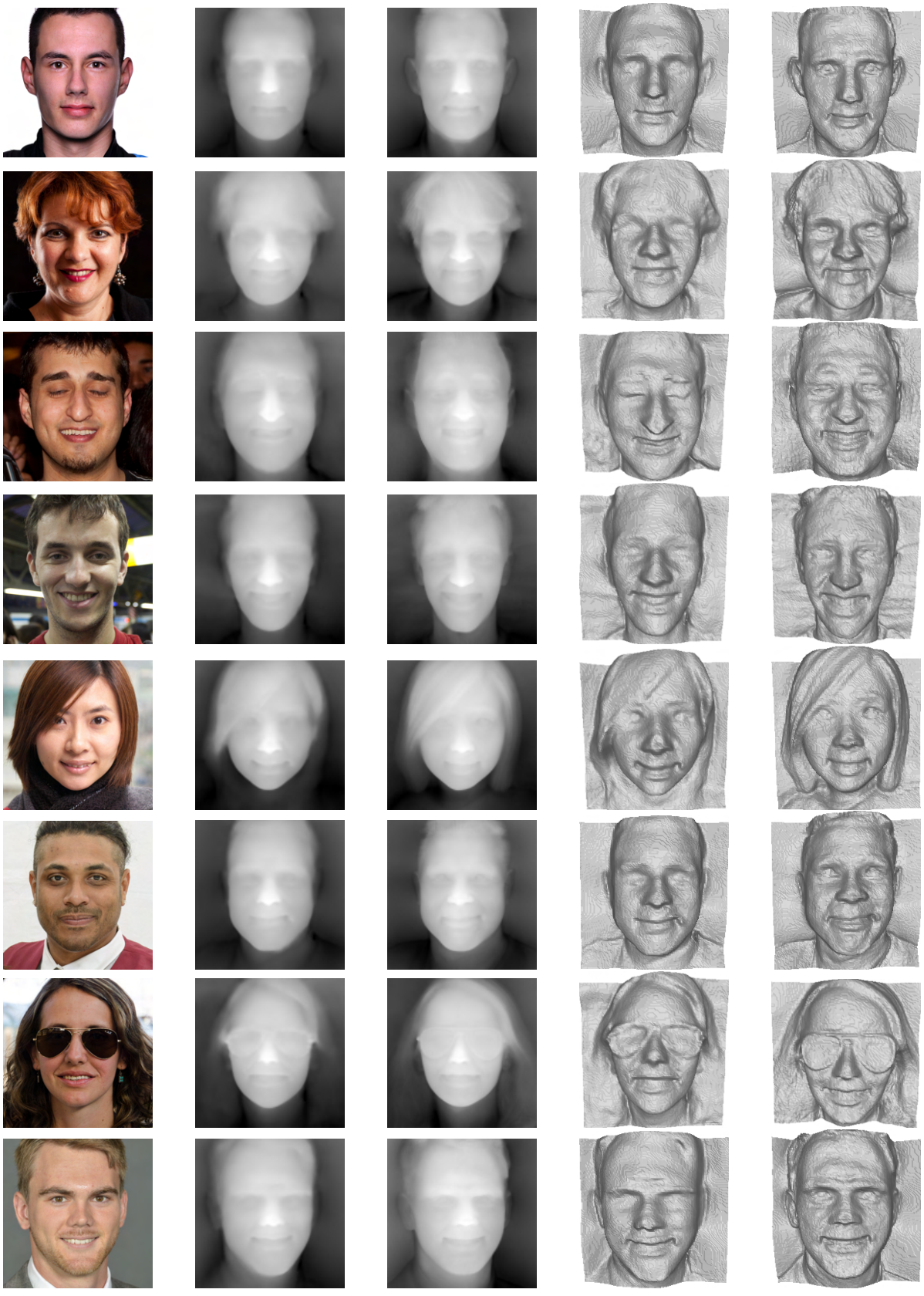


Figure 4.6: Visual evaluation results on real "in-the-wild" data from "prosopo" for the best UNET model. The input RGB image (a) is fed to the regressor, to output the corresponding depthmap (b). The ground truth depthmap (c) is depicted, along with its 3D reconstruction (e). The final predicted reconstructed mesh of the input face image is shown in d).

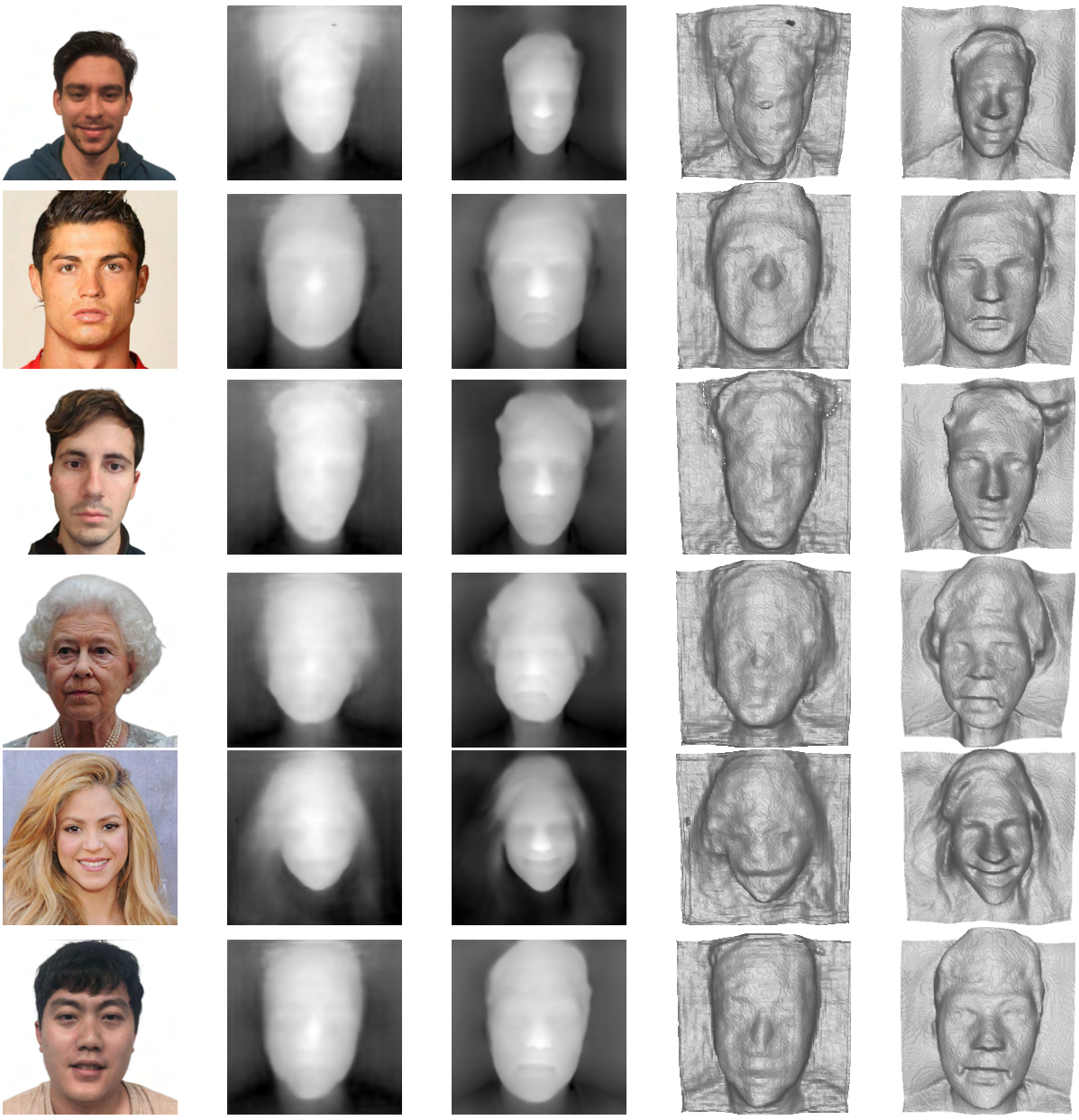


Figure 4.7: Visual results comparison for both models, on unseen "in-the-wild" real data along with some failure cases and the RVM implementation. The input RGB image (a) is fed to the regressor, to output the corresponding depthmap (b) for MiDas architecture and c) for Attention-UNET. There is no ground truth depthmap, just the predicted reconstructed face mesh for MiDas (d) and Attention-UNET (e) respectively.

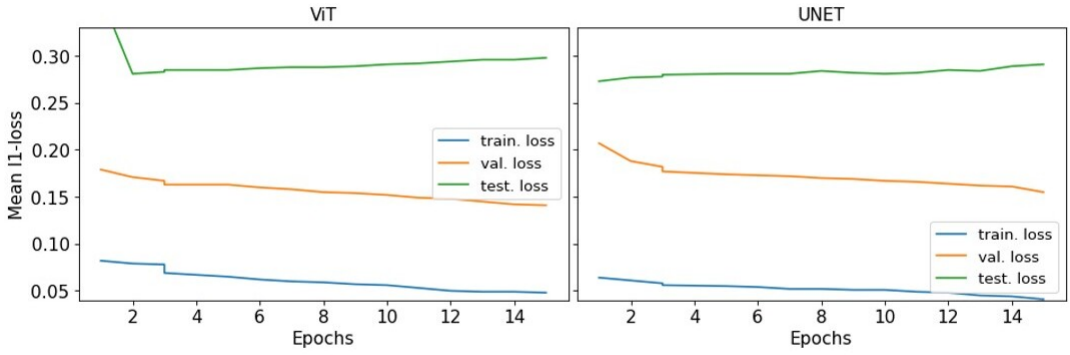


Figure 4.8: Loss vs epochs graph for both models. The testing, validation and training losses are plotted for both model architectures. The mean l1-loss over the entire set is reported, for the predicted latent codes. Both plots have the same scale, to enable comparison also among different model architectures (ViT at left or UNET at right).

Figure 4.9: Visual evaluation results on synthetic EG3D validation data for both ViT (red color) and UNET (blue color) models. The input RGB image, is the target/ideal(green color) image, whereas the generated images for each model prediction, are indicated by either red (ViT) or blue (UNET) frame around the image. The first 3 columns, compose example batch #1, whereas the last 3 columns are example batch #2.

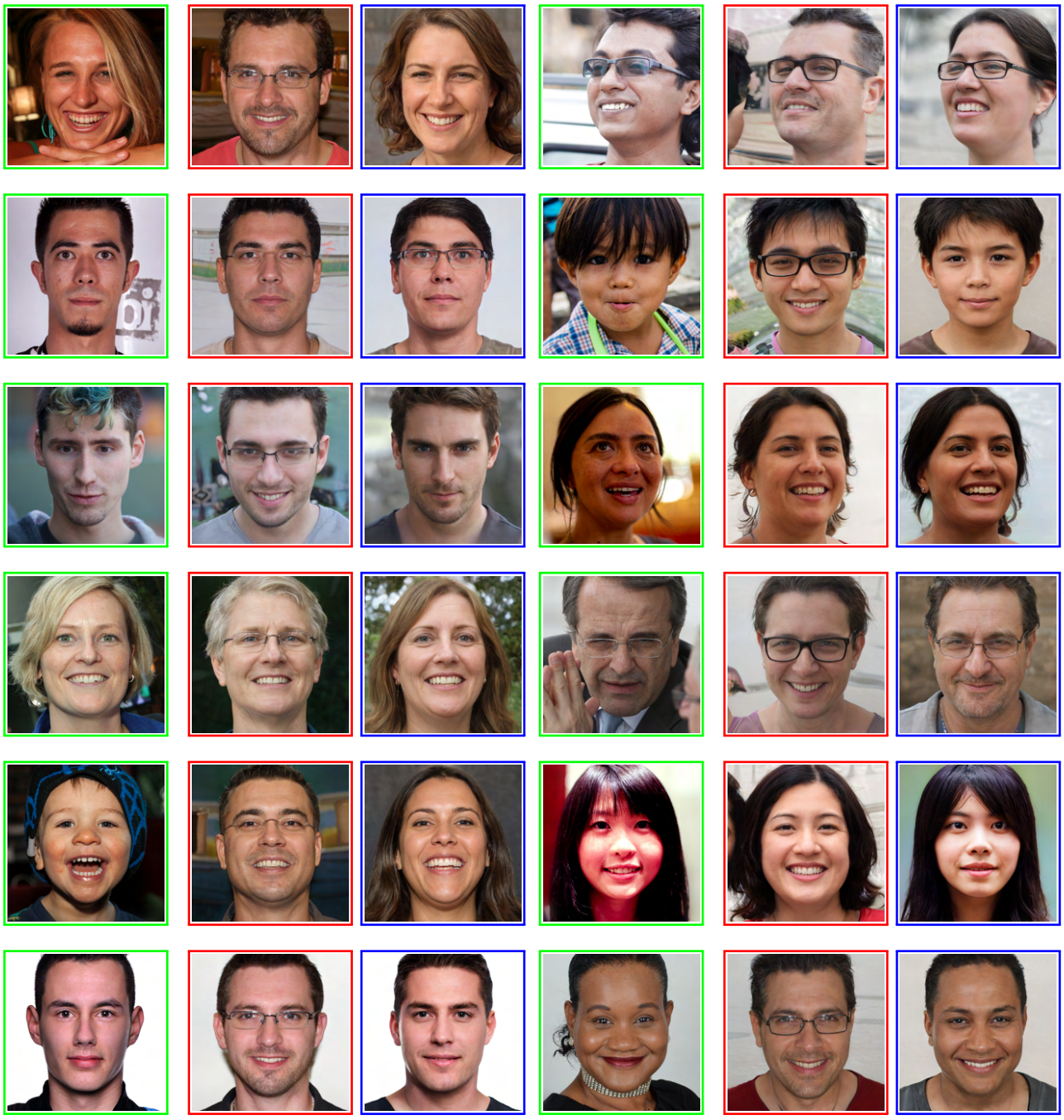


Figure 4.10: Visual evaluation results on real "in-the-wild" images for both ViT (red color) and UNET (blue color) models. The input RGB image, is the target/ideal(green color) image, whereas the generated images for each model prediction, are indicated by either red (ViT) or blue (UNET) frame around the image. The first 3 columns, compose example batch #1, whereas the last 3 columns are example batch #2.

Chapter 5

Discussion

Given an input face RGB image, the aim of both tasks, it inherently reconstruct the depicted face. For the depth regresison task, this is achieved by predicting the depthmap. As seen in the previous sections, 2 main architectures are leveraged, as main regressor architectures.

MiDaS achieves good reconstruction results, and estimates the face structure pretty accurately. Even for cases, that the background might be overlapping or merging with the foreground face, MiDaS disentangles features related to each and recovers correctly the depth around the silhouette. Nevertheless, MiDas cannot reconstruct high-frequency details and the resulting depthmaps always seem distorted. Expression specific details such as the wrinkles, or the eyebrows, are not modelled at all. The general structure of hair, the neck region and the chin is reconstructed accurately, however facial details around the mouth, eye and ear region, are not modelled robustly. MiDaS can only be used as a corase face reconstructor, as it cannot reconstruct high-quality facial details.

On the other hand Attention-UNET, performs better than the latter architecture.Mouth region and hair geometry are modelled accurately, however hair specific details (such as the hair complexity of a ponytail or a non-uniform hair look) cannot be modelled robustly. The ear region, in contrast to the MiDaS model, is partially reconstructed, however for "in-the-wild" images, the ear region geometry is not modelled in detail. The models seems to reconstruct even some wrinkle details, along with the eyebrow region. Achieving such reconstructions based on a single-view image, indicates a model, that can disentangle features from the background and foreground, whilst modeling the main facial features correctly.

The performance is comparable to models such as DECA [13], and even though the same level of wrinkle geometry reconstruction is not achieved, our model can also model hair geometry and any accessory geometry such as hat or glasses. Another baseline, LOLNerf [39], that can also model hair geometry and face accessories, achieves visually better performance. Other models, such as ImAvatar[63] and NHA [17] , visually perform better and can as well model hair geometry, nevertheless they are achieving reconstruction based on video inputs, instead of single-view images.Finally, other baseline models, such as UnSup3D [57], that models similar geometry, or RingNet [44], which leverages depth predictions, achieve visually comparable (if not worse) results, to our model. Quantitative baseline evaluation is not covered here. Overall our method, yields good performance on coarse face reconstruction, and in some cases, even detailed reconstruction is achieved. A final remark about our method, is that is does not reconstruct smooth nose geometry, as it is trained on EG3D data of ground-truth meshes of people with sharp and long noses (inherits the EG3D "nose" property) (see Appendix A.8).

Overall, Attention-UNET performs very robustly and can be directly used as an initial off-the shelf coarse face geometry reconstructor. The model enables also accurate appearance reconstruction, as the high quality face reconstruction, enables better texture rendering. As for the latent code regression task, by predicting the code and feeding it in the EG3D generator, a mesh can be reconstructed from the generated outputs. Both models, predict latent codes, that yield perceptually similar results with the target images, only on the validation

set. ViT performs better than UNET in the validation set, however the latter achieves slightly better evaluation results. Both models have comparable performance, which is not adequate on real face images. Fine-tuning the models with a large size of real training data, could resolve the overfitting of both models to the training distribution. Both models capture the main semantic features of each face, such as the skin color, the eye color, the hair length, the gender, the age and facial hair. Even though, in some instances of real faces, our models can even capture the specific expression, overall the model cannot reconstruct robustly the entire face identity.

There is no common baseline for this method, as the main purpose is to achieve perfect latent code reconstructions. Once, the regressor predicts correctly the latent code, for each face image, then EG3D appearance and geometry reconstruction quality is met. An accurate latent code EG3D predictor, could enable fast reconstruction from single-view images with EG3D quality standard. Our regressor, could potentially replace the GAN inversion process, required to project input images, to EG3D's latent space. Predicting accurately the corresponding latent code of a human face, could enable other applications such as person-specific editing or expression control. Both model architectures, do not achieve robust performance in the evaluation set and cannot be deployed, as part of a face reconstruction application.

Chapter 6

Future Work

Both frameworks, related to the depth and latent code regression task, could be further developed. As part of the testing pipeline, the depthmap regressor, could further fit the 3D geometry mesh to obtain the textured version. This is an extra short optimization task, that is performed during testing, for recovering a textured 3D mesh. As this, affects inference time, an end-to-end differentiable pipeline could be proposed, that directly regresses the 3D textured mesh of a human face, from a single face image. Instead of minimizing, only the depthmap loss, between the predicted and ground truth depthmaps, we can jointly optimize for the image reconstruction loss (between image and render). By leveraging the Pytorch3D end-to-end renderer, the framework, can learn to predict additionally to depth, an appropriate texture for the 3D mesh. During training stage, the extracted mesh is fed to the renderer (given correct camera parameters as input), to produce the final image render, that is compared to the input image. The proposed idea, is similar to the idea behind the DECAthlon pipeline described in Appendix A.7. Another technique, called double depth estimation [31], could potentially yield better dense face depth results. It can be implemented on existing pretrained depth prediction networks such as the 2 models presented, in our work (MiDaS or Attention UNET), to improve performance by generating high-resolution depthmaps with fine-grained details. This can be achieved, through optimizing the pretrained model, by merging estimations in various resolution and patch sizes. The basic assumption of this method, is that high-resolution image inputs, recover high-frequency details (e.g face wrinkles), whereas lower-resolution image inputs recover a better overall scene representation (e.g face overall shape). Hence, by using our pretrained network as the main depth prediction model, an additional merging network could be trained, to have the best of both worlds. As for the latent code regression task, to improve performance on real ”in-the-wild” images, further training is required. This means that a high number of ground truth latent codes, need to be generated using the PTI-inversion. Fine-tuning on a large sample of real data, can boost performance, to achieve similar results to the validation set. Achieving better results on the latent code regression model, could lead to achieving face reconstruction in ”real-time” by leveraging the regressor, along with the EG3D generator. A fast face reconstruction, could lead to animating the face, by reconstructing every single frame. Predicting intermediate latent codes, means that by perturbing this code, could lead to various different synthetic results. Recovering the latent code, of a person from a single input RGB image, could lead to editing semantic features of this person, such as the hair, skin color, age etc. A very interesting research area, could be latent code interpolation, which could lead to person-specific expression control.

Chapter 7

Conclusion

Single-shot 3D face reconstruction is still a challenging task, that we tried to address by presenting 2 main regression-based frameworks, whilst leveraging the generative power of EG3D. The firs method regresses dense depth values from an input RGB face image, by optimizing an l1 depthmap loss. For both network architectures, the objective is optimized by training on synthetic face images, generated by EG3D. The MiDaS model, achieves coarse geometry reconstruction that is lacking details in the ear, mouth and eye region, but successfully disentangles the face region and silhouette from its background. Attention UNET performs better and achieves detailed face reconstruction, for both real and synthetic images, by modelling hair, neck and mouth region geometry accurately. The obtained depthmaps, capture fine-grained facial details, such as eyebrows, cheek region and sometimes wrinkles. Overall, the obtained depth regression model, can reconstruct facial details, as well as hair geometry, at low inference speed, directly from single-view images. Our second method, regresses intermediate EG3D latent codes, directly from input face images. Both model architectures UNET and ViT are train by optimizing an l1-loss and fail, to robustly predict latent codes for the corresponding faces. The model performance, is robust in synthetic face images, as the main semantic face features are captured by the models. However due to the training distribution overfit, both models perform poorly with real ”in-the-wild” data. Finally, we present the ”prosopo dataset” consisting of 1600 real ”in-the-wild” images and their corresponding depthmaps, EG3D latent codes, 3D face meshes. This could become a common evaluation tool for 3D face reconstruction methods from single-view images.

Appendix A

Appendix

A.1 Appendix 1

This appendix, describes the process to acquire the evaluation/test dataset for the depthmap and latent code regression. The evaluation dataset is named ”prosopo”, meaning ”face” in Greek. It consists of 1600 manually selected FFHQ images, from in-the-wild images of real human faces. The dataset, includes input RGB images and their corresponding ground truth depth maps. The dataset is acquired by implementing the PTI-inversion method on EG3D, by using these images as input.

FFHQ images are of larger size (width=1024, height=1024), than the generator's expected input size (width=512, height=512). This leads to resizing the input images, to the appropriate size. However, the inversion results, lead to misaligned reproduced images. An example, of the inversion quality from implementing naive resizing as a preprocessing step, is seen in the Figure A.1b below. To correctly invert FFHQ images or in general images in-the-wild, the appropriate camera pose estimation of the image is required. Also, instead of naively resizing the image, the EG3D preprocessing method is implemented.

(a) Target Input Image

(b) Misaligned Inversion Results by naive resizing

(c) Aligned Inversion Results-by EG3D preprocessing step

Figure A.1: Example in-the-wild image and corresponding PTI-inversion results

This involves a number of steps:

- Aligning cropping in-the-wild image 1500 (width=1500, height=1550)
- Re-aligning to 1024 center cropping to 700
- Resizing to 512

This sophisticated pre-processing of the image, yields ideal PTI inversion results. An example of the aligned inverted image is depicted in Figure A.1c

A.2 Appendix 2

This Appendix, consists of visual examples of input images and reproduced images, from a given pivot latent code. The initial images, are projected to the EG3D latent space, by optimizing an LPIPS reconstruction loss with an added noise regularization term [25]. This inversion technique yields the pivot code, which is then fed to the original EG3D generator, to synthesize the target image, along with the depth map and shape of it. In general, the smaller the visual difference between the input and reproduced image, the more reliable are the evaluation quantitative results of the depth and code regression models. Specifically, for the latent code regression task, the pivot latent codes, do not always yield exactly the same target image. During evaluation of a latent code regression model, the EG3D generator is not fine-tuned (during testing stage), hence the ground truth latent codes of the ”prosopo” dataset, are just the pivot latent codes from the initial inversion step of the PTI method. However, the pivot latent codes, yield almost perfect results, proving the validity of our evaluation dataset. Specifically, 2 visual similarity metrics are leveraged to back this claim. A cosine similarity and LPIPS [61] metric is used to evaluate the close-to perfect reconstruction results, between the target/input image and the synthesized/reproduced/generated image. The results are reported in Table A.1 for the entire evaluation dataset. Some visual examples between the target and generated image are also depicted in Figure A.2 below.

	Cosine similarity	LPIPS
N = 1600	0.99	0.1

Table A.1: The table reports the cosine similarity and LPIPS results for the entire dataset (N=1600 images), between the input images and the generated images (based on the pivot latent code)

(a) (b) (c) (d)

(e) (f) (g) (h)

Figure A.2: Visual examples between target images (a, b, c, d) from prosopo dataset and the respective generated images from EG3D given pivot latent code (used as gt) (e, f, g, h).

As observed in example e) and f), the pivot latent code, yields perfect reconstruction results, around the face region, however it fails to reconstruct some detailed information regarding clothing and accessories. Nevertheless, detailed clothing or background information (e.g c) are not important to be reconstructed for our task.

A.3 Appendix 3

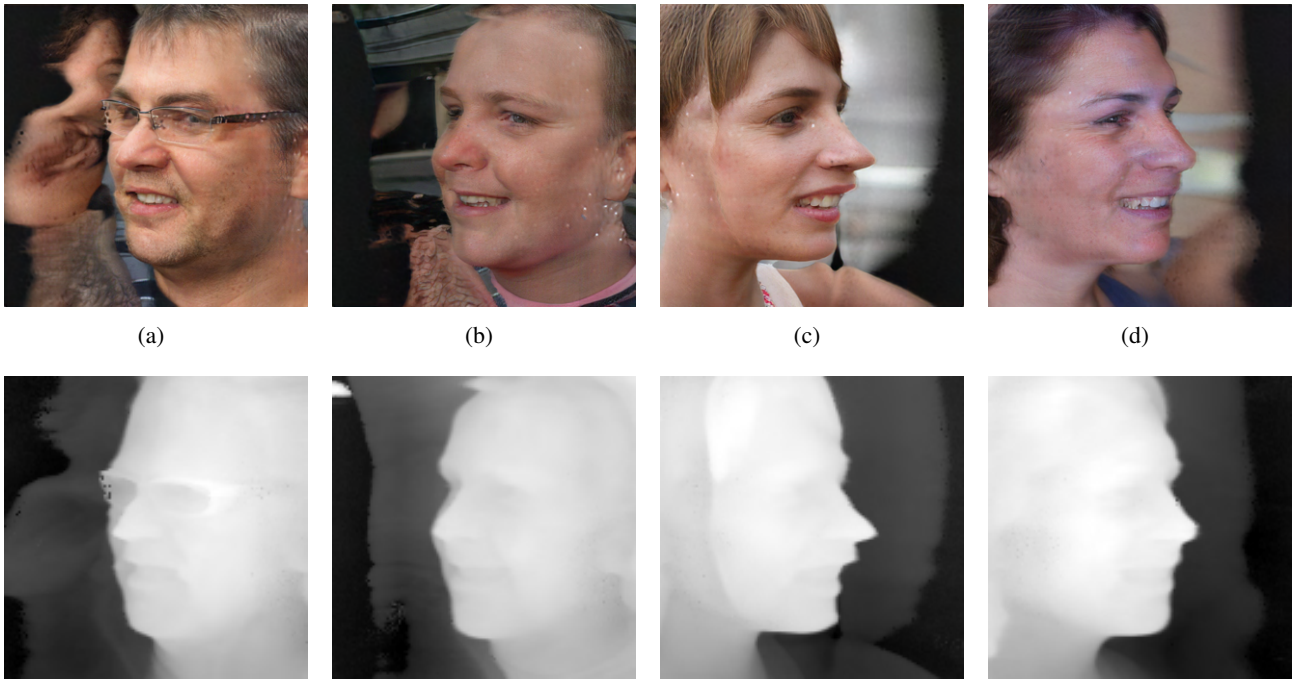
EG3D generates a synthetic face image and can change the face pose. This refers to the location and orientation of a face (e.g yaw, pitch or roll) with respect to the camera coordinate system. IN an attempt to make the trained models, more robust, side-view face images can be included in the training data. Even though EG3D synthesizes most face poses without distortions, we have observed that during a rotation of the face from the anfas position $> |70°|$, EG3D generates distorted results. The inclusion of the side-view images, is very selective, in order to avoid most of the distorted results and distill distorted information in the training process. Some distorted results are visualized in Figure A.3 below.

A.4 Appendix 4

A.4.1 fs_vid2vid Experiments

1. Download fsvid2vid to analyze the results on a synthetic image. use StyleGAN to produce a synthetic image of a human and use it as a reference image in the video to video framework. ($vid2vid\_inference.ipynb$)

2. Create a customized dataset called faceForensics, based on a driver video of your face. ($create\_dataset.ipynb$)



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure A.3: Visual examples of distorted generated images from EG3D for extreme face angles. The left most columns (a,b) show synthetic distorted examples at face rotation angles of $+72^{\circ}$ (assuming 0° at face zero pose). The right most columns (c,d) show synthetic distorted examples at face rotation angles of $+80^{\circ}$. The respective depthmaps for all distorted examples are shown in the second row (c,f,g,h)

3. Assess the results and their realism, by implementing the driver video sequence on the reference image of the synthetic human. Different driver videos and reference images are used, to evaluate the results.

Challenges:

- The initial step of extracting the facial keypoints (69-dlib landmarks) using OpenPose is detrimental to the overall performance of the fs_vid2vid framework. The use of high quality video of a human's front face, leads to better performance.
- Appropriate reference image should be used, of a person in zero pose, showing mainly the front face and kneck. The human face features should be clear in the image, to exrtact effectively the facial keypoints and ensure robust performance of the framework.

A.5 Appendix 5

A.5.1 VOCA Experiments

1. Download and install the entire VOCA model, after downloading the appropriate constituent components of it. Make sure to register to the FLAME website and the VOCA website to download the pretrained respective models. ($voca.ipynb$)
2. Record or provide an audio sequence in the .wav format. Then based on this audio signal and given a mesh template (a 3D avatar mesh (.obj)), VOCA, generates a video animation of the 3D avatar, vocalizing the audio signal ($voca.ipynb$)
3. The output animation video generated with VOCA, is later merged with the initial audio, for the final output.This assists in evaluating better the overall performance of the model. Specifically, assessing the mouth movement realism of the avatar, based on the audio sequence. As well as, evaluating the audio and avatar synchronization and overall matching.

VOCA presents the full pipeline of audio-to-3D avatar. To accommodate the needs of the project and produce a text-to-3D avatar model, text can be synthesized to speech via a TTS (Text to Speech) model, and used as audio input to the VOCA model. For the proof of concept, the project focuses on converting an audio signal to appropriate 3D avatar meshes with realistic facial expressions based on the input (audio signal)

A.6 Appendix 6

A.6.1 DECA Experiments

1. Download and install the entire DECA model, after downloading the appropriate constituent components of it. The FLAME model is substantial for predicting a 3D blend shape based on a 2D image of a face. Also the pretrained DECA model, improves the 3D mesh predictions even for images of unseen faces, with increased detail
2. Provide a 2D image of an unseen face. The experiments took place for images of real people, as well as synthetic people (produced by StyleGAN)
3. From the 2D face, FLAME produces a 3D blend shape of the face (coarse) and the texture of the face is extracted (resulting to our texture albedo map). The lightning, pose, expression, face 3D shape and texture all are connected to network parameters, leveraged by the DECA model. The initial textured coarse 3D face shape, is then combined with the displacement map, to produce the final detailed 3D face shape.

4. DECA outputs the displacement map, the texture, the depth map and a final 3D avatar mesh of the given face. There is the option of having the mesh textured or untextured.

5. DECA outputs a 3D avatar mesh based on a 2D single image of a human face. However this 3D mesh can be further stylized and change expressions. DECA offers also expression transfer based on a target expression (2D image of a human face with different expression than the initial 2D human face image) ($deca.ipynb$). It tackles the inverse rendering topic, which is quite a challenge in the field of computer vision. Based on a single face image, an avatar needs to be reconstructed.

Challenges:

- The textured 3D avatar is not very realistic. Comparing to the 2D image and the 3D avatar result, it is clear that there is still a long way to reach good levels of realism and texture matching for the inverse rendering problem.
- The expression transfer works well for the blend shape result, however the textured version is not achievable

DECA is a model that focuses entirely on generating 3D avatars based on a single image of a human face. Improving the results from DECA and producing realistic 3D avatar even for synthetic images, is groundbreaking. The results from DECA can be further utilized, by combining the expression transfer with some samples from StyleRig, for better expression fine-tuning. The resulting mesh template can be extended even to VOCA, where a realistic 3D human avatar mesh can be used as joined input with the audio signal.

Another approach could be to generate the 3D avatar animation with VOCA and then alter the facial expressions for specific frames, depending on the sentimental background of the audio, using DECA expression transfer, along with StyleRig.

A.7 Appendix 7

A.7.1 End-to-End Detailed Textured Avatar Enhancer - DECAthlon

Name inspiration: Like most athletes that participate in the DECATHLON competition, our model faces key difficult challenges. Decathlon atheletes are set to overcome major challenges to achieve their goal. Likewise DECAthlon is a model that aims to tackle one of the most challenging tasks in computer vision - the task of inverse rendering - by leveraging the existing DECA model (Siggraph 2021). This is a classic inverse rendering task with DECA model as its backbone.

1. Download and install the entire DECA model and all its prerequisites as detailed in($DECAthlon.ipynb$)

2. Download the CelebA dataset or VoxCeleb2 dataset, to train the DECAthlon model. CelebA consists of images of Celebrity faces, whereas VoxCEleb2 contains lots of videos of various celebrities. Both datasets are an available source of data, however the VoxCeleb2 has the major advantage that multipole frames for the same celebrity are available, for the various videos for each celeberity. As part of the DECAthlon model, a single image is given as input and a 3D avatar is reconstructed, with aid of DECA. The face details of this 3D avatar, depicted in 2D, are enhanced by passing it through a image enhancement NN, to improve 3D avatar realism.

3. Besides the CelebA or VoxCeleb2 dataset, StyleGAN can be used to generate random samples of synthetic faces, to train the DECAthlon model.($stylegan.ipynb$)

4. Train the end-to-end renderer/image enhancer to produce more robust 3D avatars with improved textures and detailed shapes. To achieve this the following pipeline is implemented and trained end-to end. For differentiability of our model, pytorch3d is used as our main renderer.

The overall pipeline is:

Ground Truth Image → [DECA] → Flame params, textures → [Pytorch3d] → [textured FLAME render] → [Image enhancer] → Realistic Image

And the training loss is just the discrepancy between the GT image and the final render (realistic output face image). An overview of the pipeline is shown in Figure A.4.

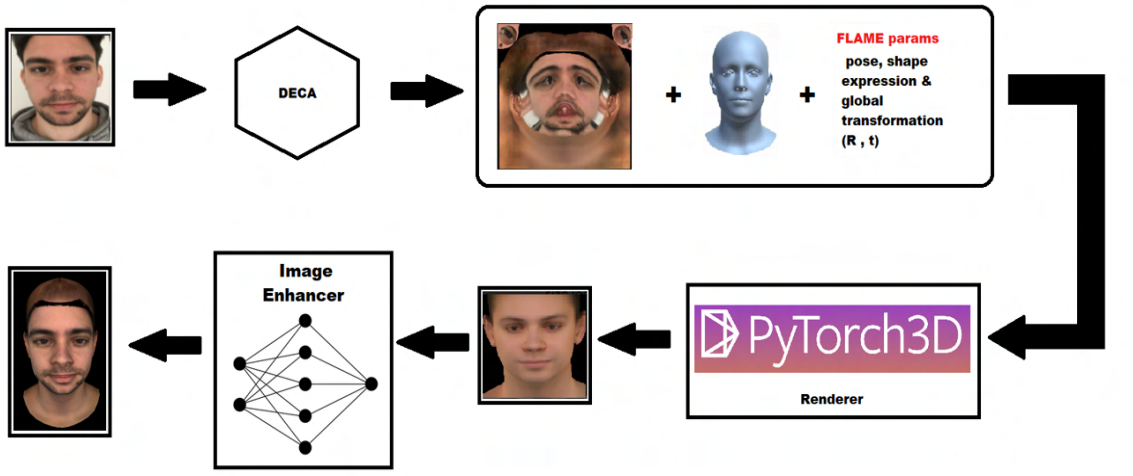


Figure A.4: Overall pipeline of DECAthlon model. The model is fed with a face image, from which DECA extracts, the FLAME parameters, along with the blend shape avatar and the texture map. The pytorch3D renderer reconstructs the textured 3D avatar face, which is used as an 2D input to the image enhancer network (IEN). The IEN outputs an improved render, that aims to align as closely to the initial input face image, which serves as the ground truth of the model. The model training loss, is the discrepancy between the GT image and the predicted realistic image render.

A.7.2 Details

1. Input 2D image, needs to have the same background with the output realistic render image. Take for example the picture in the model pipeline. The input has a white background, whereas the output has a black background. For robust training, the training loss needs to be very precise.

   Solution: Preprocessing step of removing background or converting it into white background. The face is cropped using the MTCNN face detector and enlarged to match exactly the size of the GT image face. The alignment is not exact and this might influence the results.

2. Middle pipeline should work flawlessly with matching of DECA output output .obj file.

A.7.3 Discussion

As discussed in the meeting, the first step is to ensure that the middle part of the pipeline works correctly. This means that we want to ensure that given an input face image, we extract the DECA faces, which are then parsed into the pytorch3D renderer, yielding similar results. A small sanity check for this, it could be to overfit these 2 pipeline components with a single image, until the training loss is zero. The choice of the training loss could be:

1. A perceptual similarity loss metric termed **lpips** (Learned Perceptual Image Patch Similarity). This is a loss based on visual differences of the 2 images. The prediction and reference images are fed into a

trained network that recognizes even the smallest differences between image patches. The metric is based purely on deep features and it can be applied to any architecture directly on the activation of the layers.

2. **VGG loss** is another metric based on perceptual similarity, rather than pixel-wise losses. In essence, it is defined as the euclidean distance between the feature representations of a reconstructed/predicted image and a ground truth/reference image. [20]

A.7.4 Implementation

1. Get DECA results (uv texture maps, displacement maps)
2. Optimize pytorch 3D renderer using only DECA results (no retraining of decamodel.tar)
3. FLAME parameters are used, as part of the rendering and are also optimized (FLAME model is retrained through the 3D rendering)
4. Compare the image render with the initial image and optimize for a single subject (train with multiple frames of same subject)
5. Multiple flame parameters and texture maps, but single displacement map (aggregated or random map from all video frames of the single subject)

A.7.5 Fit mesh

This set of experiments refers to fit a FLAME mesh to a textured avatar render. The DECA results are used to initialize the FLAME parameters and the texture UV map. As a result the initial rendered mesh corresponds to the reconstructed 3D avatar view from a single image. The mesh is optimized by changing the FLAME parameters related to pose, expression, etc, as well as the UV texture maps, to closely align with the ground truth image. The pixel-wise loss is the L-1 loss, given by the final mesh render and the ground truth face image. To optimize the mesh various experiments have been carried out:

- Fit mesh with single frame and overfit until the final render is very similar to the GT image.
- Fit mesh with multiple frames (taken from a video sequence)
- Fit mesh and optimize for FLAME parameters related to the face shape, expression and geometry
- Fit mesh to optimize for FLAME params related to the texture, such as the tex code (which is parsed in the FLAMETex model and produces the UV map), and determines the final texture map
- Fit mesh by directly optimizing the UV texture map
- Fit mesh by a combination of the above methods, by optimizing face and uv simultaneously for single or multiple frames, and for a variety of hyperparameters (hyperparameter tuning is applied to all experiments)

Results for each experiment can be shown below, however the optimal fitting has not been yet found. The results for each optimization experiment can be found in the Appendix.

Possible Improvements:

1. Improve preprossessing step - better render-GT image alignment, for better mesh fitting

To ensure that this works smoothly, the issue - with the discrepancy between the DECA image output and the obj. face visualization - needs to be addressed. This uses a different rasterizer I think. But not sure...there is a different result for the FLAME output tex code which results to a non detailed texture map, comparing it with the DECA texture map output.

A.7.6 Evaluation - Results Comparison

Fitting the mesh is proven to be quite challenging. The alignment of the face with the picture is required. This can be achieved by appropriate rendering of the avatar mesh face, using the correct extrinsic and intrinsic parameters. During Pytorch3D rendering the intrinsic parameters for a Perspective Camera are required, to achieve the correct face and image alignment. This can result to better texture optimization by centering the texture on the appropriate face location. Another important step is the silhouette optimization. The preprocessing of the input image is substantial, in order to remove the background of any image robustly and optimize correctly the face silhouette. RVM is the method used to extract transparent background images and optimize the silhouette accordingly. Even with the preprocessing step the silhouette optimization results, were not ideal. Most Images depict not only the facial area, but the shoulder region as well. This makes the task of silhouette optimization particularly complex. As a result, another preprocessing step is implemented, of cropping the image region tightly around the face, to improve the silhouette optimization. Even with an improved silhouette, the results are not ideal. The texture optimization works fine, however the mesh fitting overall, does not lead to a solid reconstruction of the target mesh. Results of the final rendered mesh and the optimization process can be found online and in the Appendix.

A.8 Appendix 8

EG3D meshes all commonly have long noses. This long nose property is also inherited to our depth regressors, that estimate relative depth values for the nose region, that yield similar characteristics.

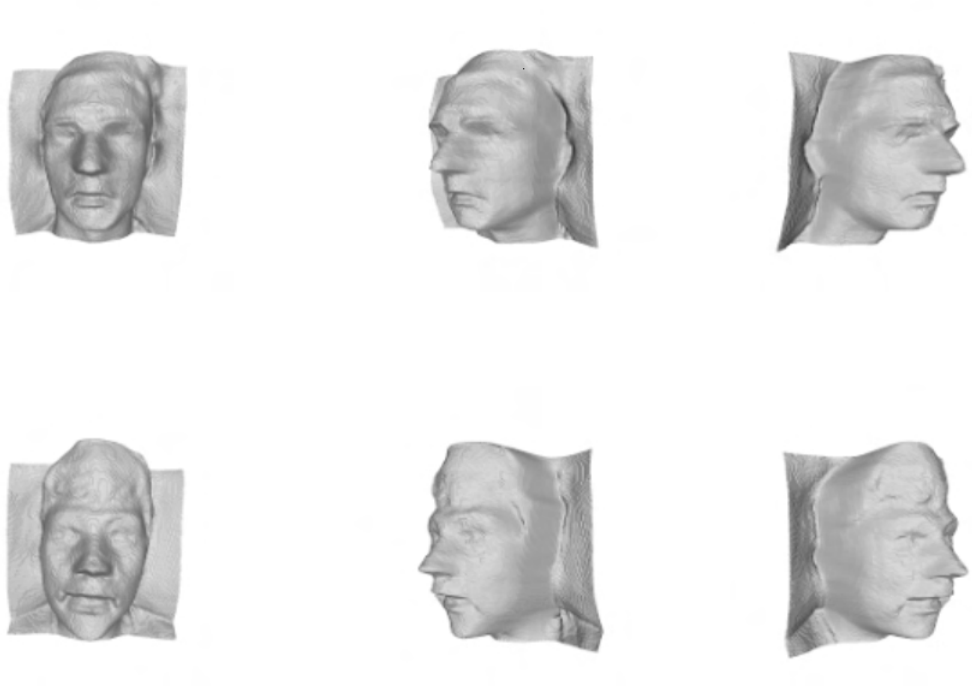


Figure A.5: Long nose property inherited from EG3D. Viewing the face predicted mesh from different FOV, the long nose property is visualized.

Bibliography

[1] depth2mesh. https://github.com/sfu-gruvi-3dv/deep_human.

[2] Google reasearch - vision transformer.

[3] Remove image background - removebg. https://www.remove.bg/.

[4] Victoria Fernandez Abrevaya, Adnane Boukhayma, Philip H.S. Torr, and Edmond Boyer. Cross-modal deep face normals with deactivable skip connections. 2020.

[5] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces, 1999.

[6] James Booth, Anastasios Roussos, Allan Ponniah, David Dunaway, and Stefanos Zafeiriou. Large scale 3d morphable models. *International Journal of Computer Vision*, 126:233–254, 4 2018.

[7] James Booth, Anastasios Roussos, Stefanos Zafeiriou, Allan Ponniah, and David Dunaway. A 3d morphable model learnt from 10,000 faces.

[8] Egor Burkov, Ruslan Rakhimov, Aleksandr Safin, Evgeny Burnaev, and Victor Lempitsky. Multi-neus: 3d head portraits from single image with neural implicit functions. 9 2022.

[9] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3d generative adversarial networks.

[10] Anpei Chen, Ruiyang Liu, Ling Xie, Zhang Chen, Hao Su, and Jingyi Yu. Sofgan: A portrait image generator with dynamic styling. *ACM Transactions on Graphics*, 41, 2 2022.

[11] Daniel Cudeiro, Timo Bolkart, Cassidy Laidlaw, Anurag Ranjan, and Michael J Black. Capture, learning, and synthesis of 3d speaking styles.

[12] Bernhard Egger, William A. P. Smith, Ayush Tewari, Stefanie Wuhrer, Michael Zollhoefer, Thabo Beeler, Florian Bernard, Timo Bolkart, Adam Kortylewski, Sami Romdhani, Christian Theobalt, Volker Blanz, and Thomas Vetter. 3d morphable face models – past, present and future. 9 2019.

[13] Yao Feng, Haiwen Feng, Michael J. Black, and Timo Bolkart. Learning an animatable detailed 3d face model from in-the-wild images. *ACM Transactions on Graphics*, 40, 7 2021.

[14] Baris Gecer, Stylianos Ploumpis, Irene Kotsia, and Stefanos Zafeiriou. Ganfit: Generative adversarial network fitting for high fidelity 3d face reconstruction. 2 2019.

[15] Stamatios Georgoulis, Konstantinos Rematas, Tobias Ritschel, Mario Fritz, Luc Van Gool, and Tinne Tuytelaars. Delight-net: Decomposing reflectance maps into specular materials and natural illumination. 3 2016.

[16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. 6 2014.

[17] Philip-William Grassal, Malte Prinzler, Titus Leistner, Carsten Rother, Matthias Nießner, and Justus Thies. Neural head avatars from monocular rgb videos. 12 2021.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 12 2015.

[19] Yasamin Jafarian and Hyun Soo Park. Self-supervised 3d representation learning of dressed humans from social media videos. 3 2021.

[20] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[21] James Kajiya and Brian von herzen. Ray tracing volume densities. *ACM SIGGRAPH Computer Graphics*, 18:165–174, 07 1984.

[22] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. 10 2017.

[23] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks.

[24] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. 12 2018.

[25] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2019.

[26] Alexandros Lattas, Stylianos Moschoglou, Baris Gecer, Stylianos Ploumpis, Vasileios Triantafyllou, Abhijeet Ghosh, and Stefanos Zafeiriou. Avatarme: Realistically renderable 3d facial reconstruction "in-the-wild". 3 2020.

[27] Tianye Li, Timo Bolkart, Michael J. Black, Hao Li, and Javier Romero. Learning a model of facial shape and expression from 4d scans. volume 36. Association for Computing Machinery, 11 2017.

[28] Shanchuan Lin, Linjie Yang, Imran Saleemi, and Soumyadip Sengupta. Robust high-resolution video matting with temporal guidance, 2021.

[29] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Transactions on Graphics*, 38, 7 2019.

[30] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery.

[31] S. H.Mahdi Miangoleh, Sebastian Dille, Long Mai, Sylvain Paris, and Yagız Aksoy. Boosting monocular depth estimation models to high-resolution via content-adaptive multi-resolution merging. 2021.

[32] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. volume 12346 LNCS, 2020.

[33] Joel Ruben Antony Moniz, Christopher Beckham, Simon Rajotte, Sina Honari, and Christopher Pal. Unsupervised depth estimation, 3d face rotation and replacement. 3 2018.

[34] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention u-net: Learning where to look for the pancreas, 2018.

[35] Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3d face model for pose and illumination invariant face recognition. pages 296–301, 2009.

[36] Rene Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44, 2022.

[37] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. 3 2021.

[38] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d.

[39] Daniel Rebain, Mark Matthews, Kwang Moo Yi, Dmitry Lagun, and Andrea Tagliasacchi. Lolnerf: Learn from one look. 11 2021.

[40] Alexander Richard, Michael Zollhoefer, Yandong Wen, Fernando de la Torre, and Yaser Sheikh. Meshtalk: 3d face animation from speech using cross-modality disentanglement. 4 2021.

[41] Daniel Roich, Ron Mokady, Amit H. Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. 6 2021.

[42] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. volume 9351, 2015.

[43] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Hao Li, and Angjoo Kanazawa. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. volume 2019-October, 2019.

[44] Soubhik Sanyal, Timo Bolkart, Haiwen Feng, and Michael J Black. Learning to regress 3d face shape and expression from an image without 3d supervision.

[45] Jo Schlemper, Ozan Oktay, Michiel Schaap, Mattias Heinrich, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention gated networks: Learning to leverage salient regions in medical images, 2018.

[46] Andreas Steiner, Alexander Kolesnikov, , Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

[47] Sicong Tang, Feitong Tan, Kelvin Cheng, Zhaoyang Li, Siyu Zhu, and Ping Tan. A neural network for detailed human depth estimation from a single image. 10 2019.

[48] Ayush Tewari, Mohamed Elgharib, Gaurav Bharaj, Florian Bernard, Hans-Peter Seidel, Patrick Pérez, Michael Zollhöfer, and Christian Theobalt. Stylerig: Rigging stylegan for 3d control over portrait images.

[49] Ayush Tewari, Michael Zollhöfer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Pérez, and Christian Theobalt. Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. 3 2017.

[50] Justus Thies, Mohamed Elgharib, Ayush Tewari, Christian Theobalt, and Matthias Nießner. Neural voice puppetry: Audio-driven facial reenactment. 12 2019.

[51] David Tingdahl and Luc Van Gool. A public system for image based 3d model generation, 2011.

[52] Evie van der Spoel, Maarten P. Rozing, Jeanine J. Houwing-Duistermaat, P. Eline Slagboom, Marian Beekman, Anton J M de Craen, Rudi G J Westendorp, and Diana van Heemst. Siamese neural networks for one-shot image recognition. *ICML - Deep Learning Workshop*, 7, 2015.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. volume 2017-December, 2017.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. An image is worth 16*16 words: transformers for image recognition at scale. volume 2017-Decem, 2017.

[55] Ting-Chun Wang, Ming-Yu Liu, Andrew Tao, Guilin Liu, Jan Kautz, and Bryan Catanzaro. Few-shot video-to-video synthesis. 10 2019.

[56] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling.

[57] Shangzhe Wu, Christian Rupprecht, and Andrea Vedaldi. Unsupervised learning of probably symmetric deformable 3d objects from images in the wild. 11 2019.

[58] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, and Jing-Hao Xue. Deep learning for single image super-resolution: A brief review. 8 2018.

[59] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models. 5 2019.

[60] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23, 2016.

[61] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.

[62] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. 7 2018.

[63] Yufeng Zheng, Victoria Fernández Abrevaya, Marcel C. Bühler, Xu Chen, Michael J. Black, and Otmar Hilliges. I m avatar: Implicit morphable head avatars from videos. 12 2021.

[64] Jun Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. volume 2017-October, 2017.