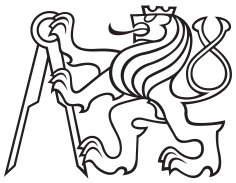
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

3D human model reconstruction and automatic rigging from a monocular video

Bc. David Otgonsuren Rico

Supervisor: Ing. Davide Castellucci
Field of study: Open Informatics
Subfield: Artificial Intelligence
May 2024

Acknowledgements

I would like to thank my family and thank my supervisor Ing. Davide Castellucci for the support, guidance and encouragement throughout this thesis.

I am also grateful to DataVision s.r.o. for granting me the thesis topic and the necessary resources, and everyone at ČVUT that helped me during my academic journey.

Declaration

I declare that this work is all my own work, and I have cited all sources I have used in the bibliography.

Prague, 20. May 2024

Abstract

This thesis investigates the generation of high-fidelity 3D animatable clothed human models from a single monocular video. Whereas existing machine learning methods achieve direct reconstruction, they lack the mesh quality of state-of-the-art methods for static scenes. This work tries to solve this issue by: First analysis of leading reconstruction methods in neural implicit representations for general-purpose 3D scenes is done. Applying these methods and fine-tuning them for human generation. Investigating and applying rigging on the generated meshes. Implementing a complete pipeline for generating a rigged human mesh from a video.

This research has the potential to improve the efficiency and accuracy of creating 3D human models for various applications, including virtual reality and animation.

Keywords: 3D-reconstruction, human, machine learning, automatic-rigging, NeRF, SDF

Supervisor: Ing. Davide Castellucci DataVision s.r.o.,

Abstrakt

Tato práce se zabývá generováním kvalitních lidských oblečených 3D modelů z monokulárního videa. Existují metody strojového učení které tento problém řeší přímo, ale nedosahují kvality modelů pro statické scény. Tato práce se snaží vyřešit tento problém následujícími kroky: Prvně, analýza nejlepších metod strojového učení pro 3D rekonstrukci. Aplikování těchto metod a jejich úprava pro 3D rekonstrukci člověka. Analýza a aplikace metod automatického mapování kostry na 3D model člověka. Implementace kompletního procesu pro generování 3D modelu člověka s kostrou z videa.

Tento výzkum má potenciál pro zlepšení efektivity a přesnosti pro tvorbu 3D lidských modelů pro různé aplikace, včetně virtuální reality a animace.

Klíčová slova: 3D rekonstrukce, člověk, strojové učení, automatické mapování kostry, NeRF, SDF

Překlad názvu: 3D rekonstrukce člověka a automatické mapování kostry z monokulárního videa

Contents

Figures

Tables

ctuthesis t1606152353

Chapter 1

Introduction

Reconstructing 3D animatable human models from videos holds immense potential for virtual reality, animation, and video game development. Whereas existing methods address parts of this challenge, they often prioritize novel-view synthesis over mesh quality or struggle to match the performance of state-of-the-art techniques for static scenes. This thesis proposes a two-step approach to overcome these limitations.

The first step leverages the neus-facto [59] method for surface reconstruction, incorporating a custom process to generate masks to isolate the human subject within the video. Reconstructing surfaces from single-view (monocular) videos has been a longstanding problem in computer vision and computer graphics. Recent advancements in neural surface reconstruction have yielded superior results compared to traditional approaches. This work investigates several methods, ultimately selecting neus-facto for its performance on the evaluation datasets. Particular emphasis is placed on generating a precise mask for the human figure within the video sequence and a custom method is used for this purpose.

The second step involves automatic rigging and skinning of the generated mesh by Neural Blend Shapes [29]. Leveraging the technique specializing in human characters in T-pose can effectively avoid skinning artifacts with standard general techniques. Using a predefined skeletal structure gives the models ability to follow mocap data.

An end-to-end pipeline is implemented using a script taking as input a video of person that stands still in a predefined pose, the video captures the

person from various angles and outputs a rigged mesh of the person in the video along with an example animation following mocap data.
To enhance user interaction, a web application was developed using the Python Flask framework. This application offers functionalities such as video upload, start of the end-to-end pipeline, and final download of the results. The application is intended for deployment on a powerful remote computer within the network, ensuring adequate computational resources for the training and pre-processing stages.

Part I

3D Human Reconstruction

Chapter 2

Preliminaries

2.1 Neural Radiance Fields NeRFs

Neural Radiance Fields (NeRF) [35] is a recent machine learning method for novel view synthesis. Given only a set of images of an object it can render novel views from other directions. Many new methods build upon this as well as 3D reconstruction methods.
A radiance field is a continuous volumetric function representing the 3d scene. It takes as an input the 3d position and the 2d viewing direction, and it outputs the corresponding volume density and directional emitted color.

$$F(x, d) = (c, \sigma) \tag{2.1}$$

where $x = (x, y, z)$ is the spatial location, $d = (\theta, \phi)$ is the viewing direction, $c = (r, g, b)$ represents the color and σ is the volume density, representing the probability that this point is occupied by objects in the scene.
NeRF represents this function by approximating this function with two MLPs a coarse and a fine version.

The training set consists of a collection of images with the corresponding camera poses. In each training iteration, a batch of pixels is randomly sampled among all possible pixels of all possible training images.

- For every sampled pixel a ray is created $r(t) = o + td$ where o is the

camera origin position and d is the camera direction

- Points are sampled along the ray getting the set of points N_c. With the stratified sampling approach the interval $[t_n, t_f]$, where t_n and t_f are near and far bounds, is partitioned into N evenly space bins and then one sample is drawn uniformly at random from each bin 2.2.

$$t_i \, \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)\right] \tag{2.2}$$

- These points are mapped to a higher dimensional space via a positional encoding 2.3. Mapping the inputs to a higher dimensional space using high-frequency functions before passing them to the network enables better fitting of data that contains high-frequency variation [35].

$$\gamma(p) = (sin(2^0\pi p), sin(2^0\pi p), ..., sin(2^{L-1}\pi p), sin(2^{L-1}\pi p)) \tag{2.3}$$

  The encoding function is applied separately to all values of $\mathbf{x}$ the spatial location and $\mathbf{d}$ the direction unit vector. And L is set separately for $\mathbf{x}$ and $\mathbf{d}$.

- Every sampled point along the ray is processed by the Coarse MLP, giving the occupancy for each point 2.1. The color output of the Coarse MLP is discarded.

- Hierarchical sampling is done along the ray with the information of occupancy from previous sampling. More samples are put where the occupancy is higher and vice versa The reasoning is that sampling should be ideally done on space that is occupied by an object. To achieve this the volume rendering equation 2.5 is rewritten as

$$\hat{C}_c(r) = \sum_{i=1}^{N_c} w_i c_i, \; w_i = T_i(1 - exp(-\sigma_i\delta_i)) \tag{2.4}$$

  Normalizing these weights as $\hat{w}_i = w_i / \sum_{j=i}^{N_c} w_j$ produces a piecewise-constant PDF along the ray. Using inverse transform sampling a second set is sampled from this distribution. The fine network is evaluated on the N_f set of samples.

- The new sampling goes as input to the Fine MLP, which returns the occupancy and color.

- For each pixel/ray r, volume rendering is used for aggregating the densities and colors of all sampling points sets N_f, N_c into a single RGB color $\hat{C}(r)$ 2.5.

$$\hat{C}(r) = \sum_{i=1}^{N} T_i(1 - exp(-\sigma_i\delta_i))c_i, \text{ where } T_i = exp(-\sum_{j=1}^{i-1} \sigma_j\delta_j) \tag{2.5}$$

  where $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples.

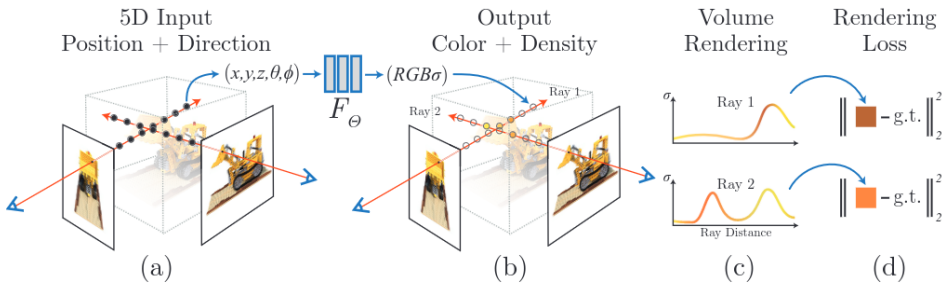


Figure 2.1: The NeRF volume rendering and training process. Image sourced from [35]. (a) shows the casting of two rays for two specific pixels, with also the generation of the samples. (b) shows the computation of densities and colors at the sampling points using NeRF MLP(s). (c) is a graphical representation of the volume rendering aggregation of the densities and colors of the rays samples for computing the rays colors. (d) illustrates the computation of the loss between estimated and ground truth pixel color(s), respectively

- Finally, the loss is computed as the Mean Squared Error (MSE) between computed colors $\hat{C}(r)$ and ground truth colors $C(r)$ among all the pixels in the batch for the coarse and fine network 2.6.

$$\mathcal{L} = \sum_{r \in R} \left[\|\hat{C}_c(r) - C(r)\|_2^2 + \|\hat{C}_f(r) - C(r)\|_2^2 \right] \tag{2.6}$$

  where R is the set of rays in each batch and $C(r)$, $\hat{C}_c(r)$ and $\hat{C}_f(r)$ are the ground truth, coarse volume predicted, and fine volume predicted RGB colors for ray r respectively.

Minimizing this error across multiple views encourages the network to predict a coherent model of the scene by assigning high volume densities and accurate colors to the locations that contain the true underlying scene content. The overview of the training and rendering process can be seen in figure 2.1.

The training is done per scene, the MLPs learn a specific scene and are not generalizable to others. And the training for an individual scene with such an approach can take a long time. The trade-off is between previous methods is time versus space where NeRF only needs memory space for the weights of the network which can be even less than the image itself [35], but methods using voxel grids can take many GBs of memory for a single scene. On the other hand these 3D voxel grid methods might need only a few minutes to process a dataset whereas a single optimization with NeRF typically takes around 100-300k iterations to converge on a single NVIDIA V100 GPU (about 1-2 days) [35].

Figure 2.2: Reference image on the left followed by mesh obtained with NeuS and lastly by NeRF. The amount of noise in the NeRF model shows the disadvantage of using purely volumetric rendering for surface reconstruction. Image taken from [49].

NeRF is not a great solution when task is 3D reconstruction. A problem of NeRF and its variants, however, is the question of how an isosurface of the volume density could be defined to represent the underlying 3D geometry. Current practice often relies on heuristic thresholding on the density values. Due to insufficient constraints on the level sets, however, such surfaces are often noisy and may not model the scene structures accurately [38].

2.2 Neural Signed Distance Function (SDF) representation

The previous section discussed limitations of Neural Radiance Fields (NeRF) for high-quality surface extraction due to its reliance solely on a volumetric density field. This chapter explores NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction[49], which addresses this challenge.

Neus incorporates the volume rendering approach 2.1 to learn a neural Signed Distance Function (SDF) representation. This approach combines the strengths of both methods: achieving accurate surface representation with a neural SDF and enabling robust network training even for scenes with sudden depth changes (a benefit of volume rendering). Figure 2.2 visually compares the two methods. Unlike in novel-view synthesis, the goal in this approach is to reconstruct the surface S of a 3D object from a set of images capturing the object from different viewpoints. NeuS accomplishes this by representing the scene with two functions:

- $f : \mathbb{R}^3 :\rightarrow \mathbb{R}$ This function maps a 3D spatial position (x) to its signed distance from the object's surface. Positive values represent points

outside the object, while negative values indicate points inside.

- $c : \mathbb{R}^3 \times \mathbb{S}^2 \rightarrow \mathbb{R}^3$: This function encodes the color associated with a specific point x in 3D space and a viewing direction $v \in \mathbb{S}^2$.

Both f and c are implemented using Multi-Layer Perceptrons. The surface S of the object is ultimately defined by the zero-level set of the SDF function $f(x)$. Points where $f(x)$ equals zero represent the exact surface of the 3D object.

$$S = \{x \in \mathbb{R}^3 | f(x) = 0\} \tag{2.7}$$

With a probability density function $\phi_s(f(x))$, called S-density, where $f(x)$, $x \in \mathbb{R}^3$, is the signed distance function and $\phi_s(x) = se^{-sx}/(1 + e^{-sx})^2$, commonly known as the logistic density distribution. Intuitively, the main idea of NeuS is that, with the aid of the S-density field $\phi_s(f(x))$, volume rendering is used to train the SDF network with only 2D input images as supervision. Upon successful minimization of a loss function based on this supervision, the zero-level set of the network-encoded SDF is expected to represent an accurately reconstructed surface S, with its induced S-density $\phi_s(f(x))$ assuming prominently high values near the surface.

To this goal the volume rendering is done similarly as in NeRF see equation 2.4. Choosing the appropriate weight function w is crucial as to build an appropriate connection between the output colors and SDF. The requirements for the weight function are that it must be unbiased and occlusion aware. An unbiased weight function $w(t)$ guarantees that the intersection of the camera ray with the zero-level set of SDF contributes most to the pixel color. The occlusion-aware property ensures that when a ray sequentially passes multiple surfaces, the rendering procedure will correctly use the color of the surface nearest to the camera to compute the output color.
The weight function that satisfies these properties is defined as,

$$w(t) = T(t)\rho(t), \text{ where } T(t) = exp(-\int_0^t \rho(u)du) \tag{2.8}$$

where ρ is the opaque density, which is the standard counterpart to volume density in classic volume rendering. The opaque density function is defined as

$$\rho(t) = max\left(\frac{-\frac{d\phi_s}{dt}(f(p(t)))}{\phi_s(f(p(t)))}, 0\right) \tag{2.9}$$

With similar approximation as in NeRF the pixel color of a ray with n sampled points $p_i = o + t_i v | i = 1, ..., n, t_i < t_{i+1}$ is computed as

$$\hat{C} = \sum_{i=1}^{n} T_i \alpha_i c_i, \text{ where } T_i = \prod_{j=1}^{i-1}(1 - \alpha_j) \tag{2.10}$$

and α_i is the discrete opacity values defined by

$$\alpha_i = max\left(\frac{\phi_s(f(p(t_i))) - \phi_s(f(p(t_{i+1})))}{\phi_s(f(p(t_i)))}, 0\right) \tag{2.11}$$

With this it's possible to minimize the difference between the ground truth colors and rendered colors without any 3D supervision while having a connection between the rendered colors and the SDF. Detailed derivation of all the formulas in this section can be found in the original paper [49].

2.3 Multi-resolution hash encoding

A significant drawback of NeRF and neural Signed Distance Function (SDF) methods is their lengthy training times per scene. This arises from the computational cost of training and evaluating fully connected neural networks, the core architecture used in these techniques.
Recent research presented in "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding" [36] tackles this inefficiency challenge for NeRF and other related methods. This approach significantly reduces the number of floating-point operations and memory accesses required during training by introducing a novel type of input encoding. This innovation improves training efficiency without compromising output quality to a significant degree.
The key aspect of this method lies in selecting an appropriate hash table size. This size dictates the trade-off between performance, memory usage, and final reconstruction quality. Larger hash tables generally lead to higher quality reconstructions but come at the cost of slower performance.

A small neural network is augmented by a multiresolution hash table of trainable encoding parameters θ whose values are optimized through stochastic gradient descent. These are arranged into L levels, each containing up to T feature vectors with dimensionality F. Each level is independent and conceptually stores feature vectors at the vertices of a grid. The resolution of which is chosen to be a geometric progression between the coarsest and finest resolutions $[N_{min}, N_{max}]$:

$$N_l = \lfloor N_{min} b^l \rfloor,\ b = exp\left(\frac{\ln N_{max} - \ln N_{min}}{L-1}\right) \tag{2.12}$$

In a single level l the input $x \in \mathbb{R}^d$ is scaled to the resolution of the level N_l before being rounded down and up. These rounded values of x represent a voxel with 2^d integer vertices in Z^d. Each corner of the voxel is mapped to an entry in the given feature array for the level. For coarse levels with fewer

than T parameters the mapping is 1:1. For finer levels a hash function is used to index into the feature array, basically being treated as a hash table. A spatial hash function is used:

$$h(x) = \left(\bigoplus_{i=1}^{d} x_i \pi_i \right) \mod T \tag{2.13}$$

where $\bigoplus$ represents the bit-wise XOR operation and π_i are unique large prime numbers.
Lastly the feature vectors at each corner are d-linearly interpolated according to the relative position within the voxel, the interpolation weight is $w_l = x_l - \lfloor x_l \rfloor$.

This process is done independently for each level making parallelization of this process easier. The interpolated feature vectors are concatenated along with auxiliary inputs $\xi \in \mathbb{R}^E$ which produces the final encoded input $y \in \mathbb{R}^{LF+E}$. See the visual representation of this process in 2D in figure 2.3. This new encoding replaces the positional encoding 2.3 seen in the NeRF section.

A hash collision occurs when two different 3D coordinates map to the same index in the hash table. While "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding" [36] does not explicitly address collision resolution, the approach demonstrates the ability to represent scenes faithfully even with collisions.
There are no hash collisions in coarse levels of the hash tables where there are fewer than T parameters, but in fine levels hash collisions happen progressively more frequently.
However, the paper argues that these collisions are:

- Randomly scattered: Collisions are distributed randomly throughout the space, minimizing the chance of simultaneous collisions at all levels for a specific pair of points.
- Statistically unlikely to impact all samples equally: Collisions tend to involve points with varying importance. Points on the object's surface with visible colors likely contribute more significantly to the table entries compared to empty space points that collide with them. This means "important" samples tend to dominate collisions, minimizing the impact of collisions on overall quality.

In essence, the averaging effect and varying sample importance help mitigate the impact of hash collisions in this multiresolution approach.

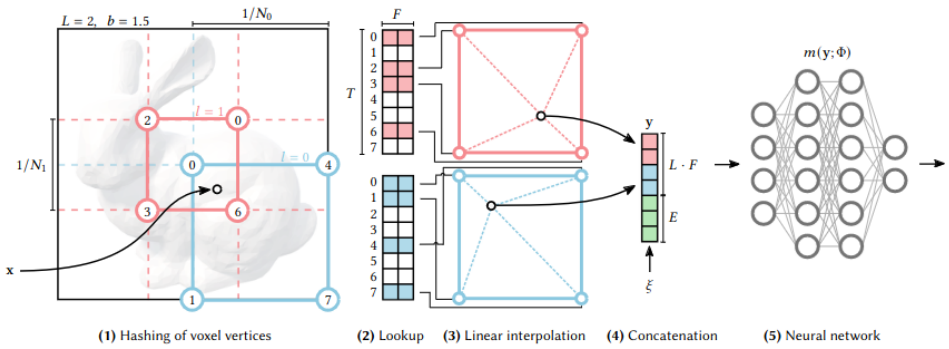


Figure 2.3: Illustration of the multi-resolution hash encoding in 2D. Image taken from [36].

Chapter 3

3D Surface Reconstruction ML Methods

In this section we will compare machine learning methods for 3D surface reconstruction of static scenes, where the methods share many or modified versions of parts discussed in the preliminaries section 2. This work tries to reconstruct 3D human meshes from a monocular video.

In this method the human subject in the video does not move and stays still in a specified pose. Therefore, we shall consider the person as a static object. Even if it might be impossible for a person to achieve complete stillness, this method intentionally tries to take a different approach from other methods which try to reconstruct a moving person in a video [17], [22], or perform novel view synthesis of a person from a video [24], [52], [31].
These methods try to take a moving person and transfer them to a canonical space and most try to approximate them to a parametric human model. This can cause issues where the transferring the human to and from the canonical space can introduce artifacts and trying to approximate clothed people or people with long hair to a parametric human model like skinned multi-person linear model SMPL [32], which is bald and nude often leads to deterioration of the clothing and hair and being merged with the skin surface of the model. Some methods can animate the model only to a pose seen in the video and those that can to arbitrary do so by either driving the model with a template mesh like SMPL or actually using a learned skinned deformation field to transform the model from a canonical pose to a non-canonical pose. However, none of these methods produces a rigged mesh as the output. This approach addresses these limitations by prioritizing the creation of a high-quality mesh followed by automatic rigging. This strategy allows for greater flexibility in using the final model across various software applications.

3.1 Neus

In this section the NeuS [49] method is briefly described. Main idea of NeuS has been already described in detail at 2.2 where the connection between the SDF and volume rendering has been made.
Just like NeRF, NeuS also uses two MLPs:

- SDF head. Instead of the volume density head.

$$f(x) = MLP_{\theta_1}(x) \tag{3.1}$$

- Color head. Just like NeRF.

$$c(x, d) = MLP_{\theta_2}(x, d) \tag{3.2}$$

The same training procedure as in NeRF is applied (see section 2.1). Given a ray $r(t) = o + td$ the same discretized volume rendering formula 2.10 is used, similar to NeRF.
Let $r_k(t)_{k=1..M}$ be the batch of pixels/rays. In each ray, let $r(t_i)_{i=1..N}$ be the sequence of samples. The overall loss $\mathcal{L}$ is computed as a weighted sum of three components.

- Color loss. The average of the MSE between ground truth and predicted ray colors.

$$\mathcal{L}_{color} = \frac{1}{M} \sum_{k=1}^{M} \|\hat{C}(r_k) - C(r_k)\|_2^2 \tag{3.3}$$

- Regularization loss. Term for the regularization of the SDF.

$$\mathcal{L}_{reg} = \frac{1}{NM} \sum_{k=1}^{M} \sum_{i=1}^{N} (\|\nabla_f(r_k(t_i))\|_2 - 1)^2 \tag{3.4}$$

- Mask loss. This loss is optional, can be added if ground truth masks are provided.

$$\mathcal{L}_{mask} = \frac{1}{M} \sum_{k=1}^{M} BCE(M_k, \hat{O}_k),\ \hat{O}_k = \sum_{i=1}^{N} T_{k,i}\alpha_{k,i} \tag{3.5}$$

  $\hat{O}_k$ is the sum of weights along the camera ray and $M_k \in 0, 1$ are the mask values, BCE stands for binary cross entropy.

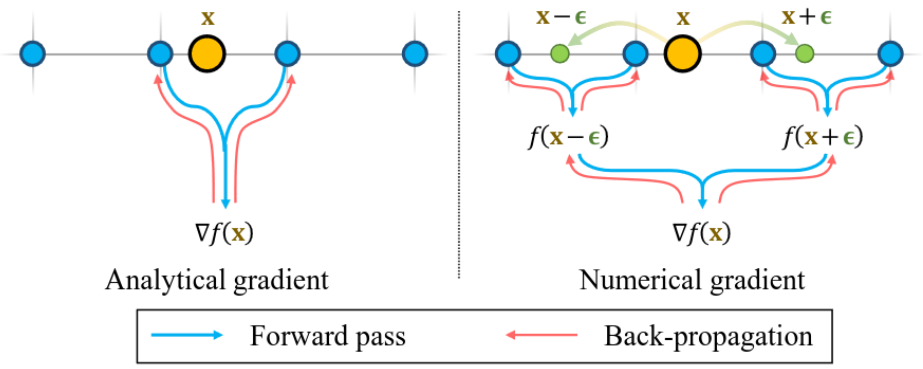


Figure 3.1: Illustration of when using numerical gradients the update of hash grid encoding happens beyond local hash grid. Image taken from [30].

3.2 Neuralangelo

This section briefly describes the novel state-of-the-art method Neuralangelo [30] for 3D surface reconstruction.
In a way Neuralangelo builds on top of Neus 3.1. It consists of an SDF head and color head as well. What Neuralangelo adds is using numerical gradients to compute higher-order derivatives, such as surface normals for the eikonal regularization [16] and a progressive optimization schedule for the hash encoding. Both of these techniques improve the hash encoded reconstruction.

3.2.1 Resolving localities in hash grids with numerical gradients

Authors of Neuralangelo propose an alternative method to computing the gradients of the hash grids. Instead of using analytical gradients which update only the local hash grids the new method of computing numerical gradients updates beyond the local hash grids which improves the detail between the regions such as smoothness. Example of this operation can be seen in figure 3.1.

A special property of SDF is its differentiability with a gradient of the unit norm. The gradient of SDF satisfies the eikonal equation $||\nabla f(x)||_2 = 1$ (almost everywhere). To enforce the optimized neural representation to be a

valid SDF, the eikonal loss is typically imposed on the SDF predictions:

$$\mathcal{L}_{eik} = \frac{1}{N}\sum_{i=1}^{N}(||\nabla f(x_i)||_2 - 1)^2 \tag{3.6}$$

where N is the total number of sampled points. To allow for end-to-end optimization, a double backward operation on the SDF prediction $f(x)$ is required. Notice the eikonal loss being essentially the regularization loss in Neus 3.4.

The de facto method for computing surface normals of SDFs $\nabla f(x)$ is to use analytical gradients [49], [58]. Analytical gradients of hash encoding w.r.t. position, however, are not continuous across space under trilinear interpolation. The feature vectors of the hash encoding of a grid size V_l for a sampled point x_i are

$$\gamma_l(x_{i,l}) = \gamma_l(\lfloor x_{i,l} \rfloor) \cdot (1 - (x_{i,l} - \lfloor x_{i,l} \rfloor)) + \gamma_l(\lceil x_{i,l} \rceil) \cdot (1 - (x_{i,l} - \lfloor x_{i,l} \rfloor)) \tag{3.7}$$

This corresponds to the description of feature vectors in 2.3. The derivative of hash encoding w.r.t. the position can be obtained as

$$\frac{d\gamma_l(x_{i,l})}{dx_i} = \gamma_l(\lfloor x_{i,l} \rfloor) \cdot (-V_l) + \gamma_l(\lceil x_{i,l} \rceil) \cdot V_l \tag{3.8}$$

When x_i moves to a different voxel in the grid the derivative changes resulting in the eikonal loss 3.6 suffering from localities. With numerical gradients the surface normals will be computed using additional 6 sampled points for x_i 2 on each axis within a step size ϵ. Resulting in the x-component of the surface normal being computed as:

$$\nabla_x f(x_i) = \left[\frac{f(\gamma(x_i + \epsilon_x)) + f(\gamma(x_i - \epsilon_x))}{2\epsilon}\right] \tag{3.9}$$

Where $\epsilon_x = [\epsilon, 0, 0]$. Other component are computed in similar fashion with appropriate $\epsilon_i, i \in x, y, z$ vector.

3.2.2 Coarse to fine optimization

In contrast to other methods, Neuralangelo utilizes a progressive optimization scheme for surface reconstruction. This approach gradually incorporates finer details into the generated surface.

- Numerical gradients are used for optimization. The step size ϵ controls how much the model updates its parameters in each iteration.

- A high ϵ value ensures consistency across larger surface areas during the initial optimization stages. This helps establish the overall shape effectively.
- Conversely, a lower ϵ value focuses on smaller areas, preventing excessive smoothing and allowing for the preservation of fine details.

The step size ϵ is not constant throughout training. It is reduced exponentially as the optimization progresses. This reduction schedule aligns with the activation of progressively finer hash grids in the model.
Neuralangelo does not activate all hash grids at the beginning of training. Instead, they are gradually activated as the step size ϵ reaches a value corresponding to their spatial resolution.
If all hash grids were active from the start, the finer grids would need to "unlearn" their initial contributions as the step size shrinks. This could lead to the loss of valuable geometric details captured at finer scales during the early stages of optimization where larger features are being established.

The total loss of the method is defined as:

$$\mathcal{L} = \mathcal{L}_{RGB} + w_{eik}\mathcal{L}_{eik} + w_{curv}\mathcal{L}_{curv} \tag{3.10}$$

where $\mathcal{L}_{RGB}$ is equivalent to 3.3, $\mathcal{L}_{eik}$ is defined in 3.6 and $\mathcal{L}_{curv}$ is the curvature loss. Curvature loss is meant to further encourage smoothing by regularizing the mean curvature.

$$\mathcal{L}_{curv} = \frac{1}{N}\sum_{i=1}^{N} |\nabla^2 f(x_i)| \tag{3.11}$$

3.3 Neus-facto

The neus-facto model is the custom model of SDFstudio [59]. SDFStudio is a unified and modular framework for neural implicit surface reconstruction, built on top of the nerfstudio framework [47]. The starting point of this method is NeuS 3.1 and Mip-nerf 360 [6].
Mip-nerf 360 model is NeRF variant that focuses on handling unbounded scenes. The method approaches this problem by tackling three main problems.

- Parametrization. Unbounded 360 degree scenes can occupy an arbitrarily large region of Euclidean space, but mip-NeRF requires that 3D scene

coordinates lie in a bounded domain. The proposed solution is scene contraction.

- Efficiency. Large and detailed scenes requires bigger and slower models and more samples. The proposed solution are proposal networks.
- Ambiguity. The problem of reconstructing the 3d model from 2d images is inherently ambiguous and can produce various artifacts. The proposed solution is distortion loss.

Scene contraction. The 3d space is re-parametrized, in order to constraint the space to a bounded domain. Such operation is called "scene contraction". The following scene contraction function $f : \mathbb{R}^3 \to \mathbb{R}^3$ is used

$$f(x) = \begin{cases} x & ||x||_2 \leq 1 \\ \left(2 - \frac{1}{\|x\|_2}\right)\left(\frac{x}{\|x\|_2}\right) & \|x\|_2 \geq 1 \end{cases} \tag{3.12}$$

The points inside the unit sphere remain unaffected and points outside it are mapped to a sphere of radius 2. When sampling a ray, the sequence of t values is generated in order to be distributed linearly in disparity, meaning that the higher is the depth then the fewer the generated samples are.

Proposal networks. The standard coarse-to-fine NeRF (see 2.4) approach utilizes two separate NeRF networks: a "coarse" one and a "fine" one. While the "coarse" NeRF predicts both RGB colors and weights (volume densities), only the weights are ultimately used for training. This strategy has an inefficiency:

- The "coarse" NeRF is unnecessarily large and computationally expensive because it predicts colors that are ultimately discarded.
- The model is trained using color information, even though only the weights are relevant for the final reconstruction.

Mip-NeRF proposes a more efficient two-network architecture that overcomes these limitations:

- Proposal Network: This network acts like a lightweight NeRF, focusing solely on predicting weights (densities). Given an initial sample $\hat{t}$, it outputs the corresponding weight $\hat{w}$.
- NeRF Network: This network remains similar to the original NeRF, predicting both weights (densities) and colors (c).

Sampling strategy.

- Initial Sample Distribution: The process starts with generating initial samples $\hat{t}$ distributed linearly in disparity.
- The proposal network takes these initial samples $\hat{t}$ and predicts weights $\hat{w}$ along with their probability density function (PDF).
- Based on this PDF, new samples t are generated using inverse transform sampling.
- The NeRF network takes the refined samples t and predicts both the final weights w and colors c.

The training strategy for the proposal network differs from the NeRF network. Instead of using color information, it leverages the knowledge from the larger NeRF network through a technique called online distillation. This approach involves training the two networks simultaneously. The well-trained NeRF network acts as a "teacher," transferring its knowledge to the smaller proposal network, which acts as a "student." The proposal network is trained to predict a probability density function distribution $\hat{w}$ that closely resembles the PDF distribution w produced by the NeRF network. This essentially teaches the proposal network to predict weights (densities) in a way that aligns with the final NeRF predictions. The proposal loss function measures the difference between the two predicted PDF histograms using bins.

$$\mathcal{L}_{prop} = \sum_i \frac{1}{w_i} max(0, w_i - bound(\hat{t}, \hat{w}, T_i)) \tag{3.13}$$

where $bound(\hat{t}, \hat{w}, T)$ is a function that computes the sum of all proposal weights that overlap with interval T:

$$bound(\hat{t}, \hat{w}, T) = \sum_{j:T \cap \hat{T}_j \neq \emptyset} \hat{w}_j \tag{3.14}$$

By minimizing this loss, the proposal network learns to generate weight distributions that are more consistent with those of the NeRF network. This online distillation approach allows the proposal network to become more effective at predicting weights, even though it's a smaller and less complex network compared to the NeRF network.

The scene contraction and proposal networks are both used in the neus-facto method. The differences being that the NeRF networks are replaced by their neural SDFs as in 2.2, the scene contraction uses the L1 norm and space is therefore contracted to cubes and not spheres and proposal networks

also utilize hash encodings.
The overall loss for neus-facto consists of the color loss $\mathcal{L}_{RGB}$, eikonal loss $\mathcal{L}_{eik}$ and an optional mask loss $\mathcal{L}_{mask}$ which are practically equal to the losses in Neus 3.1. Lastly the additional proposal loss from Mip-nerf 360 for proposal networks is added as well.

3.4 Mesh generation using Marching Cubes

This chapter discusses the marching cubes algorithm [33], a technique used to generate a polygonal mesh representing the surface of an object defined by an iso surface in this case a Signed Distance Function (SDF) field.

- Discretizing the SDF: The SDF is divided into a 3D grid with N x N x N points. Each point's value represents the distance from that point to the object's surface.
- Analyzing Cubes: The algorithm iterates through the grid, processing eight neighboring points at a time (forming a cube).
- Classifying Vertices: Each vertex of the cube is treated as a binary value:
  - 1: If the SDF value is positive (inside the surface).
  - 0: If the SDF value is negative (outside the surface).
- Combinations and Triangles: With 8 vertices, there are 2^8 (256) possible configurations for a cube. Each configuration corresponds to a predefined set of triangles that represent the portion of the surface intersecting the cube.
- Mesh Assembly: Finally, all the triangles generated from individual cubes are merged into a single, unified mesh surface. Figure 3.2 illustrates some of these cube configurations.

To determine the precise vertex location along a cube edge, the algorithm performs linear interpolation between the two SDF values at the edge's endpoints.
The normal vector for each generated vertex is calculated by interpolating the gradient (slope) of the SDF at the surrounding grid points.

This work utilizes a multi-resolution version of the marching cubes algorithm within SDFstudio. Since the NeuS SDF is a continuous function, the algorithm

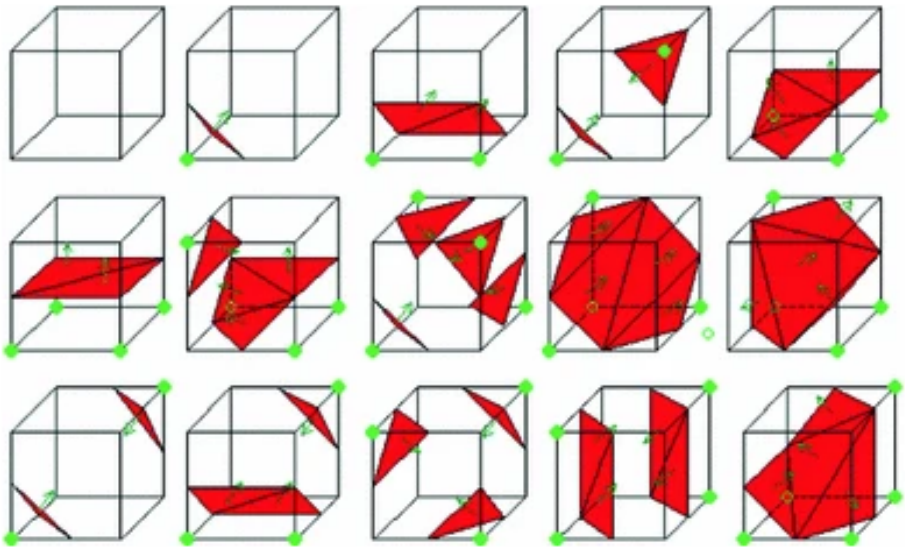


Figure 3.2: Representation of some possible cube combinations in the marching cube algorithm. Image taken from [12].

can be applied at different resolutions with varying N x N grids. This approach generally produces a finer and more detailed mesh compared to a single resolution.
The marching cubes algorithm is not limited to SDFs. It's a general-purpose method applicable to any isosurface function, where a specific threshold value defines the surface (in the case of SDFs, the threshold is zero). While the marching cubes algorithm can also be used with the NeRF volume density function (σ), an appropriate threshold value (iso-value) needs to be carefully chosen.

3.5 Texturing meshes

This section discusses how to add textures to meshes generated by SDFStudio's Marching Cubes algorithm. SDFStudio [59] does not inherently create textured meshes. To achieve this, the software utilizes the model's color rendering capabilities.

The texturing process: Input and Output

- The process takes an existing mesh as input.

- It outputs a new mesh with identical geometry but including texture information.

UV mapping

- Texture information is defined as a mapping between mesh vertices and a specific region of a texture image. This mapping is called UV mapping.
- Each mesh vertex is assigned a pair of UV coordinates (between 0 and 1). These coordinates represent the location of a corresponding pixel (texel) within a 2D texture image.
- The texture image essentially serves as a 2D representation of the 3D object's surface. This process was done using the xatlas [2] software.

Coloring the Texture Image

- The goal is to determine a color for each texel (pixel) in the texture image based on the provided color rendering of the model.
- Find the 3D point p and normal vector n corresponding to the texel t. This is achieved using barycentric interpolation with the inverse of the UV mapping.
- Cast a ray through point p with the negative normal vector -n as the viewing direction.
- Utilize the model's color rendering to determine the color c that would be observed along this ray. And set this color to the queried texel.

Computing the 3D point can be done by identifying the triangle in the texture image that encompasses the texel t. Employing the inverse UV mapping to retrieve the 3D coordinates of the triangle's vertices in the mesh. Apply barycentric interpolation to calculate the 3D location p within the triangle that corresponds to the texel t. This essentially reverses the operation performed during rendering.
Barycentric interpolation is a mathematical technique used for interpolation within triangles. In this context, it's used to find both the 3D point and normal vector associated with a specific texel based on its UV coordinates and the surrounding triangle's properties. Visualization of the barycentric representation can be seen in figure 3.3, which consists of a triangle defined by its three vertices: A, B, and C, a point P located inside the triangle,

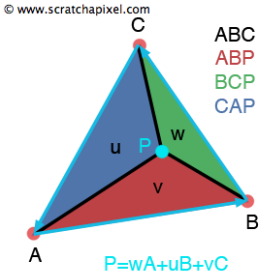


Figure 3.3: Illustration of the barycentric representation for a triangle. Image taken from `https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates.html`.

Figure 3.4: Example of a generated mesh without textures and of the same mesh after the texturing process.

barycentric coordinates for each vertex (u, v, w). These coordinates represent the weights given to each vertex in the interpolation process. They range between 0 and 1, and their sum must always equal 1 (u + v + w = 1). Calculating these weights for a texel t inside a triangle we can use the same weights to get the point P in the triangle in 3D space and other way around. The ray cast through the point P is defined as $r(t) = o + td$, where $d = -n$, $o = P - sd$ s is a scalar multiplier, this because we want the origin of the ray to lie outside the mesh and direction of the ray to face towards the mesh while passing through the point P. The near plane is set as zero, while the far plane is set as double the scalar factor s. The texturing process essentially performs the inverse of what happens during rendering. It takes information from the 3D model and uses it to determine the appropriate colors for the 2D texture image. A visual comparison of the non textured mesh can be seen in Figure 3.4.

3.6 Evaluation

This section compares the methods described in this chapter for 3D human reconstruction and novel-view synthesis. For 3D reconstruction the evaluation is done on the RenderPeople dataset [1] and for the novel-view synthesis comparison a custom dataset was created, the dataset is available for download on `https://drive.google.com/file/d/1awDaKDnSu0qn6C43eUsNPk0XFUpLYqi8/view?usp=sharing`.
The used implementations of the models are from the SDFstudio framework [59]. The models share the same hyperparameters for all the trainings. The exception being the number of iterations where for surface reconstruction on the synthetic dataset it is 50k and for novel-view synthesis it is 200k. The more important parameters that are shared are the number of levels for the hash tables $L = 16$ where the maximum resolution of a hash grid is $N_{max} = 2048$, number of rays per batch 2048. The models are optimized using the Adam optimizer [27]. For all hyperparameters used in the training such as the learning rates or size of the MLPs, can be found in the file **default_config.yml** in the attachments.

3.6.1 Surface reconstruction

To compare different methods on human mesh generation the requirement is that dataset has the ground truth mesh of the clothed human and a monocular video or image sequence of the person in a still pose from different views. For this purpose a partially synthetic dataset is created where high quality textured human scans are used to render the mesh from different views. With the rendered views as inputs the accuracy of 3D reconstruction methods between the generated and ground truth meshes can be evaluated.

The human scans are taken from the Render People dataset [1], we use 10 different human scans of various genders, skin tones and ages. Meshes are also in various poses but most of them are in the default T-pose also used in our custom videos. Example of the textured meshes can be seen in Figure 3.5. The synthetic dataset is created by normalizing the meshes, by setting them to the center of the coordinate system and scaling them to fit into a unit sphere. Then 100 different views are rendered around the mesh at a fixed position along with their masks, example of the rendered views can be seen in Figure 3.6. Camera positions are transferred to the Nerfstudio coordinate system and bounding box and initialization sphere are calculated from the vertices of each mesh individually. Each of the methods runs for

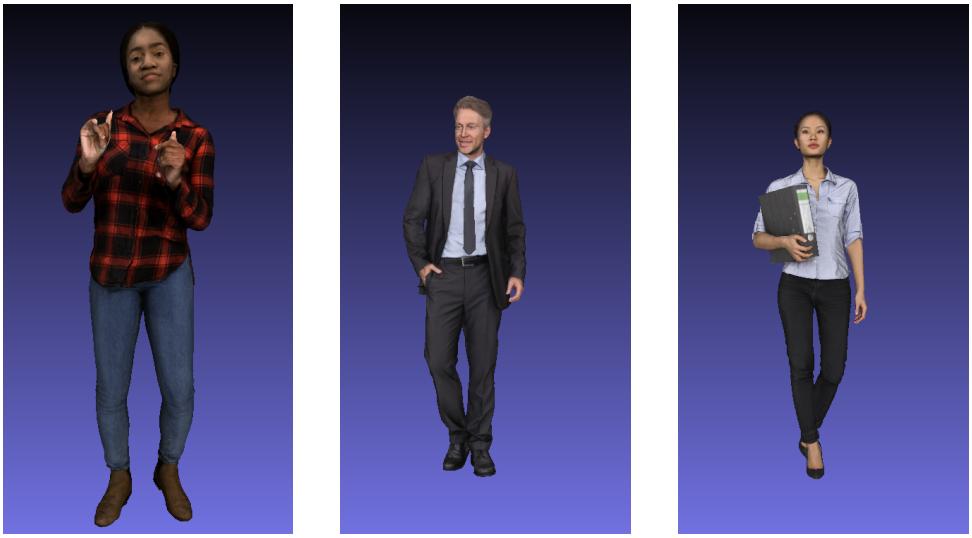


Figure 3.5: Example of the human scans from the RenderPeople dataset.

Figure 3.6: Visualization of example rendered views for a single scan.

50k iterations on each mesh. Results can be seen in the table 3.1 from which we can see that the neus-facto achieves the best results in terms of Chamfer distance. Chamfer distance is a measure to evaluate similarity between shapes. To calculate Chamfer distance for a set of shapes A and B, for each point in set find the closest point in the other set, measure distance between all pairs and average them. In this case the sets of points are vertices of the ground truth mesh $V = \{V_i \in \mathbb{R}^3 | V_i \text{ is a vertice of mesh } V\}$ where N is the number of vertices in the ground truth mesh and vertices of the predicted mesh $\hat{V} = \left\{\hat{V}_i \in \mathbb{R}^3 | \hat{V}_i \text{ is a vertice of mesh } \hat{V}\right\}$ where M is the number of vertices in the predicted mesh. The Chamfer distance can be defined as:

$$CD(V, \hat{V}) = \frac{1}{N}\sum_{i}^{N} \min_{k} d(V_i, \hat{V}_k) + \frac{1}{M}\sum_{i}^{M} \min_{k} d(\hat{V}_i, V_k) \qquad (3.15)$$

	Aliyah	Carla	Claudia	Dennis	Eric
Neus	0.073253	0.124543	0.131229	0.094924	0.124431
Neuralangelo	0.067635	0.127798	0.132934	**0.089336**	**0.122494**
neus-facto	**0.067269**	**0.127463**	**0.131135**	0.091416	0.123004

Percy	Manuel	Mei	Nathan	Sophia	Mean
0.078438	0.100412	0.126827	0.124126	0.116659	0.109484
0.066369	0.100675	0.131113	0.123846	0.115012	0.107721
0.068347	**0.098791**	**0.118981**	**0.120181**	**0.112117**	**0.105870**

Table 3.1: Evaluation of 3D reconstruction methods on the synthetic RenderPeople dataset using Chamfer distance in mm.

3.6.2 Novel view synthesis

This section introduces the evaluation of the neus-facto, Neus and Neuralangelo models for novel-view synthesis on videos containing a human subject. The models go beyond just mesh creation, it also possesses color rendering capabilities. This functionality is crucial for texturing the reconstructed meshes. Color information is also essential during training for the RGB loss function.
Compared to 3D reconstruction, creating a dataset for novel view synthesis is generally simpler. A custom real-world dataset is utilized for this purpose. The dataset consists of 5 videos capturing a human subject in a canonical T-pose while the camera rotates around them. The dataset is available for download on `https://drive.google.com/file/d/1awDaKDnSu0qn6C43eUsNPk0XFUpLYqi8/view?usp=sharing`. On average, 300 frames are extracted from each video, and 10% are used as a validation set. Each model is trained for 200k iterations on each video.

The validation set is used to calculate PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index Measure) metrics, which evaluate the quality of the synthesized novel views.

PSNR stands for Peak Signal-to-Noise Ratio. It's a metric commonly used in image processing and reconstruction tasks to assess the quality of a reconstructed image compared to an original reference image.
Peak Signal: Refers to the maximum possible value (intensity) a pixel in the image can represent. This value depends on the number of bits used to encode the image (e.g., 255 for 8-bit images).
Signal-to-Noise Ratio (SNR): This ratio compares the actual signal (the original image data) to the background noise introduced during reconstruction or transmission. A higher SNR indicates a better quality image with less noise.
PSNR calculates the difference between the corresponding pixels in the original

and reconstructed images. These differences are squared, averaged across all pixels, and then converted to the logarithmic decibel (dB) scale.

$$\mathrm{PSNR}(I, J) = 10 * \log_{10}\left(\frac{\max(I)^2}{\mathrm{MSE}(I, J)}\right) \tag{3.16}$$

where I is the original image, J is the predicted image and MSE is the mean squared error.

$$\mathrm{MSE}(I, J) = \frac{1}{mn}\sum_{1}^{m}\sum_{1}^{n}\|I(i, j) - J(i, j)\|^2 \tag{3.17}$$

where m represents the numbers of rows of pixels of the images and n represents the number of columns of pixels of the images.
A higher PSNR value generally indicates a better reconstruction quality, with perfect fidelity resulting in a PSNR of infinity (in theory). In practical scenarios, PSNR values typically range between 30 dB and 50 dB for good quality reconstructions.

SSIM stands for Structural Similarity Index Measure [51] is a quality metric to measure quality between two images. The metric is considered to be correlated with the quality perception of the human visual system, where PSNR focuses primarily on pixel-wise differences and does not necessarily reflect how well the reconstructed image preserves human-perceived details or visual quality.
SSIM compares three image characteristics of the original and reconstructed images.

- Luminance. Measures the overall brightness or intensity of the image.

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{3.18}$$

  where μ_x is

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.19}$$

  where N is the number of pixels in the image.

- Contrast. Assesses the level of distinction between different brightness levels in the image.

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{3.20}$$

  where σ_x is

$$\sigma_x = \sqrt{\left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)} \tag{3.21}$$

	apartment	office1	office2	living room	kitchen	Mean
PSNR						
Neus	32.9830	26.2393	24.7268	27.5688	30.2017	28.3439
Neuralangelo	34.3220	**26.3180**	24.7370	**27.6153**	30.1058	28.6196
neus-facto	**34.3535**	26.2080	**24.8047**	27.5919	**30.2121**	**28.6340**
SSIM						
Neus	0.9586	0.8035	**0.7456**	0.8689	0.9139	0.8581
Neuralangelo	0.9639	**0.8040**	0.7435	0.8698	0.9114	0.8585
neus-facto	**0.9640**	0.8031	0.7411	**0.8701**	**0.9147**	**0.8586**

Table 3.2: Evaluation of novel view synthesis capabilities of the 3D reconstruction methods using the PSNR and SSIM metrics.

- Structure. Evaluates how the pixels within a local neighborhood are arranged spatially.

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3} \tag{3.22}$$

where σ_{xy} is the covariance between x and y

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y) \tag{3.23}$$

The positive constants C_1, C_2 and C_3 are used to avoid a null denominator and x, y are the reference and generated image, the order does not matter since $S(x,y) = S(y,x)$. Each comparison (luminance, contrast, structure) is transformed into a separate similarity score between 0 (no similarity) and 1 (perfect similarity). These individual similarity scores are then combined into a single SSIM value, typically ranging between 0 and 1. A higher SSIM value indicates a greater structural similarity between the original and reconstructed images.

$$S(x,y) = f(l(x,y), c(x,y), s(x,y)) \tag{3.24}$$

The evaluation of the custom dataset on the validation data using the above described PSNR and SSIM metrics can be seen in Table 3.2. From the results it can be seen that the methods achieve very similar results for all the videos in the dataset, however the neus-facto method performs slightly better in both metrics than the other approaches. An example of the novel-view synthesis capabilities of the models can be seen in figure 3.7.

While the models achieve high-quality novel view synthesis, the extracted meshes from the custom dataset training exhibit some noticeable errors. These artifacts primarily manifest as protrusions on the hands and occasionally the

Figure 3.7: Example of the novel view synthesis capabilities of neus-facto on the custom dataset, original image (left), generated image (right).

face. Example of these errors can be seen in the Figure 3.9.
Several factors can contribute to mesh errors:

- Camera Pose Estimation: Inaccurate camera positions estimated by COLMAP can lead to reconstruction inconsistencies.
- Masking Issues: False negatives (missing mask regions) can introduce holes in the mesh, while false positives (including irrelevant areas) can add unwanted surface details.
- T-Pose Constraint: Requiring the subject to maintain a T-pose throughout the video likely contributes to these artifacts. Holding a T-pose for an extended period is unnatural, and hand drift is often observed (Figure 3.8 clearly shows hand drop at the video's end).

One of the potential improvements might be utilizing a more natural pose for the subject in the video recordings could alleviate hand and face deformation issues. However, this introduces a challenge: the current rigging pipeline requires a T-pose mesh as input. The extracted meshes from the custom dataset and the results of the rest of the pipeline are available for download at `https://drive.google.com/file/d/1leXTQgjrBjYtDzsKYiA4FLD3YArQr9Wh/view?usp=sharing`.

Figure 3.8: The figure shows an image where outline of a mesh is used as a foreground over an image. Notice how the hands do not correspond with the rest of the outline.

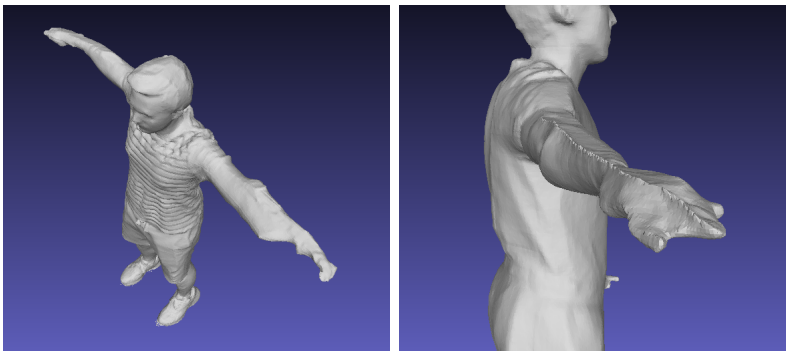


Figure 3.9: Visualization of artifacts that appear on meshes generated from the custom dataset.

Chapter 4

Foreground Segmentation Techniques for Human Subject Isolation

Several computer vision techniques leverage binary masks as additional input to separate foreground objects from the background scene. In the context of this thesis, such masks are crucial for isolating the human subject from the scene for subsequent 3D reconstruction and achieving improved quality by allowing the model to focus purely on the human subject, see figure 4.1 for visual comparison. Human segmentation approaches of a video can be broadly categorized into two main paradigms: frame-wise segmentation and segmentation with temporal information.

Frame-wise segmentation methods process each video frame independently, generating a separate mask for each frame. While computationally efficient, these methods may struggle with handling rapid motion or occlusions across frames. Conversely, temporally coherent segmentation techniques leverage temporal information from adjacent frames to produce more consistent masks. This can be particularly beneficial for handling dynamic scenes with motion blur or partial occlusions.

The selection of an appropriate segmentation technique is critical for this work due to the sensitivity of the chosen surface reconstruction method (NeuS-Facto and its variants) to inaccurate masks. Errors in the segmentation process can readily introduce artifacts into the final reconstructed mesh, compromising its quality.

(a) : First frame **(b) :** Mesh not masked **(c) :** Mesh masked

Figure 4.1: Visual comparison of a reconstruction of human mesh without the use of masks 4.1b and with 4.1c from a video 4.1a

4.1 2D pose keypoints as a segmentation prompt for initial mask

Obtaining mask of humans can be done in a variety of ways. Inspired by the recent work of InstantAvatar [23] where the creation of a human avatar for 3D novel-view synthesis needs to segment the human subject from the video. For InstantAvatar this was achieved by frame-wise segmentation with the Segment Anything model (SAM) [28].

SAM is a promptable model for image segmentation tasks, given a segmentation prompt a segmentation mask is returned. The ability to prompt the model gives it the option to adapt to a large variety of different image segmentation tasks. It provides competitive or improved results over supervised methods [28]. The types of prompts it can be given are points, bounding boxes, text or masks. SAM has three main components, the image encoder that gets the image embedding from the image. Prompt encoder handles differently sparse (points, boxes, text) and dense (masks) prompts, points and boxes are handled via a positional encoder [46], text with the text encoder CLIP [40] and masks are embedded using convolutions. Finally, the mask decoder takes the image and prompt embedding and maps them to a mask. Three masks are outputted along with their confidence score to handle the ambiguity of the prompts.

InstantAvatar [23] uses 2D human keypoints estimates as point prompts to the SAM model. The 2D human keypoints are estimated with OpenPose [8]. OpenPose is an open-source real-time system for multi-person 2D pose detection. The system takes a color image and outputs 2D locations of anatomical keypoints for each person as well as confidence scores for each of the keypoints. This approach works well for the InstantAvatar dataset where the subject in the video is always in full view and rotates in front of the camera. Another reason for the use of SAM for human segmentation is the SAM dataset includes a high number of high quality human segmentations and performs well across many different groups. Certain groups of people have been underrepresented based on gender, skin tone and age. More precisely these groups are females, people with darker skin tones and young and old people [60, 56]
In this method's approach the human keypoint estimation is done with MMPose [14] a pose estimation toolbox using the HRNet [9] method for bottom up human pose estimation.

This approach gives high quality segmentation masks even compared to the state-of-the-art methods for segmentation like Detectron [15, 53] using the famous Mask R-CNN method [20] for object detection and instance segmentation. An example visualization of the image, keypoints and mask can be seen in figure 4.2. However, there are issues when using this approach for our dataset. Unlike in the InstantAvatar dataset in ours the subject does not rotate in front of the camera but instead the camera rotates around the subject in a manner where for some frames the subject is only partially visible. In these special cases the keypoint estimation won't return anything and then the SAM segments the image without a prompt generating a mask that's not necessarily focused on a person causing large discrepancies between the frames where this occurs. An example of these failed segmentation can be seen in figure 4.3. One of main issues appears when another person appears in the video, even if for a moment the keypoints can be detected and used as a segmentation prompt. This could be resolved by restricting the video to never contain more than one person. However, this can prove quite challenging when even a momentary reflection of the subject or cameraman can fail the segmentation.

To solve this issue we propose to move from frame-wise segmentation to video object segmentation using the temporal information to make the generated masks more robust. Semi-supervised methods for video segmentation often outperform the unsupervised methods [57], but they often require an additional input such as an initial mask. We'll use the SAM prompted by human pose estimation on the first frame of the video and use it as the initial mask for semi-supervised video object segmentation method.

(a) : Original image **(b) :** Keypoints **(c) :** SAM mask

Figure 4.2: The original image 4.2a, visualized keypoints in image 4.2b, SAM mask generated using 2D human keypoints as prompts 4.2c

4.2 Semi supervised video object segmentation with X-MEM

Many semi supervised video object segmentation methods require an initial mask that they will track for the rest of the video [57]. These semi supervised methods often outperform unsupervised methods, but getting a high quality initial mask can be time consuming if done manually. For our purposes we need to get a high quality mask of the human subject in the first frame. This can be done easily with the above mentioned method of the Segment Anything Model with estimated human 2d keypoints as segmentation prompts.

Then for our method we can use the state-of-the-art method XMem: Long-Term Video Object Segmentation with an Atkinson-Shiffrin Memory Model [10]. The main difference between other video segmentation methods is that XMem uses three deeply connected feature memory stores instead of one. The Atkinson-Shiffrin memory model is a psychological model of memory that proposes that memory consists of three stores: a sensory register, short-term memory and long term memory. Inspired by this psychological model XMem incorporates multiple independent yet deeply-connected feature memory stores: a rapidly updated sensory memory, a high-resolution working memory, and a compact thus sustained long-term memory. XMem also avoids use of excessive amount of memory for long videos by adding frequently used working memory to long-term memory.

(a) : Image 1 **(b) :** Mask 1 **(c) :** Image 2 **(d) :** Mask 2

Figure 4.3: The issues of using SAM for every frame include no other people can appear in the video 4.3a,4.3b, including reflections 4.3c,4.3d

Overview of how XMem works: for the initial mask the feature memory is initialized then for every subsequent frame the memory is read from sensory, short-term and long-term memory. The readout features are used to generate a segmentation mask. Each of the feature memories is updated at different frequencies.
Long term memory gets features when the working memory reaches a predefined threshold and memory consolidation is performed. Potential candidates are the most frequently used features from the working memory, and they will be converted to long-term memory representations and are then added to the long term memory. When the long-term memory is full, obsolete features are discarded to bound the maximum GPU memory usage.
Working memory gets features from the sensory memory every r-th frame. Working memory is similar to many other feature memories in other video object segmentation methods, in this case employing a STCN network [11]. Number of frames in working memory is limited with a predefined threshold and when reached consolidating the extra frames to the long-term memory.
Sensory memory is updated at every frame and stores low-level information such as object location. The sensory memory is updated with features of a decoder. Every r-th frame a deep update is performed. Refreshing the sensory memory and discarding redundant information.

4.3 Evaluation

The TikTok dataset [21], consisting of 340 short videos featuring various people performing dances, is chosen for evaluating the performance of different segmentation methods. This dataset offers several advantages for our

evaluation purposes. Firstly, each video contains only one person, satisfying a prerequisite for our 3D human reconstruction approach. Secondly, while the dataset primarily focuses on people in motion and a static camera, not directly relevant to our method with a static subject and a moving camera, the inherent motion within the videos provides an opportunity to assess the segmentation methods' ability to handle partially occluded or blurred subjects often in fast motion, in a more difficult scenario than ours. This evaluation will allow us to identify a segmentation technique that offers a robustness to dynamic scene elements, ultimately leading to higher human mesh reconstruction on the custom dataset.

The methods compared are from frame-wise segmentation: Detectron2 [15, 53] which is a platform for object detection, segmentation, IsNet [25] and U2-Net [39] which are state-of-the-art methods for foreground segmentation techniques. From the video object segmentation methods is the previously described 4.2 keypoint guided SAM + XMem.

The evaluation metrics used are Precision, Recall, Dice Similarity Coefficient (DICE) score in case of single object segmentation equivalent to the F1 score and Intersection over Union (IoU) also known as the Jaccard index.

$$\text{IoU}(A, B) = \frac{A \cap B}{A \cup B} \tag{4.1}$$

$$\text{DICE}(A, B) = \frac{2(A \cap B)}{A + B} \tag{4.2}$$

The results can be seen in table 4.1 where the means for every metric have been calculated over all the ground truth and predicted masks. There is no method that seems obviously better than all the others as no method dominates all the metrics. In this method's use case it is important that there are no large discrepancies between the estimated mask and ground truth, since for the reconstruction with SDF methods only one completely wrong mask can put undesired artifacts in the resulting mesh. Larger error in the segmentation needs to be penalized heavily and this is what IoU emphasizes. High Recall ensures all relevant parts of the object are included for accurate classification. It is perhaps not surprising that for this metric a video object segmentation method performs better than a frame-wise segmentations approaches. Keeping the temporal information of previous frames helps XMEM with initial mask from SAM with human keypoints as segmentation prompts to be more robust than the other methods. This is why we'll choose this approach for human segmentation in our case.

(a) : Original **(b)** : GT **(c)** : Detectron2 **(d)** : U2net **(e)** : IsNet **(f)** : XMEM

Method / Metric	Precision	Recall	DICE	IoU
Detectron2	0.9603	0.9358	0.9470	0.9090
Isnet	0.9851	0.9258	0.9504	0.9130
U2net	**0.9893**	0.9379	**0.9604**	0.9286
SAM + XMEM	0.9737	**0.9423**	0.9562	**0.9358**

Table 4.1: Evaluation of different masking methods on the TikTok dataset

Part II

Automatic Rigging

Chapter 5

Automatic rigging of a mesh

In this chapter an overview of automatic rigging and animation methods is presented and chosen. While there are many methods on how to possibly animate a clothed human from a video, such as driving a human avatar from sparse RGB-D inputs [54] or a single RGB sequence [24], mesh pose transfer [48] [45] in which the pose of a source mesh is applied to a target mesh. These methods can achieve impressive results, but in our case the method has a single RGB input such as a video or image sequence, the person is in a still canonical pose, the output has to be a rigged, skinned human mesh. For these reasons we cannot consider these methods for our own use or comparison.
The rigging process traditionally requires an animator to create an animation 'skeleton' and bind it to an input mesh. The skeleton represents the articulation structure of the character, and skeletal joint rotations provide an animator with direct hierarchical control of character pose. Skinning is the process of binding the input mesh to the joint setup. A rig may consist of many joints and most should only influence certain parts of the mesh, i.e. a wrist joint should control only the wrist part of the mesh.

Automatic rigging is a long-standing problem in computer graphics. The pioneering work Pinnochio [5] proposes a template based method that automatically fits a user-provided skeleton to a target mesh and creates an animation ready rig. The method works as follows:

- It packs spheres inside the mesh and constructs a graph on their centers.
- Finds the embedding of the given skeleton into the graph.
- Refinements of the positions of the skeleton joints within the character.

- Computes the bone weights for skeleton subspace deformation.
- Use of online motion retargeting to eliminate footskate.

The requirements are to give the mesh and input template skeleton in roughly the same position and orientation. Making no differentiation between materials on the mesh makes every part of the mesh move in a 'similar' fashion, which can lead to a 'rubbery' motion. Pinnochio is an essential work in automatic rigging, but many others have followed up on this task since then, and more recently also methods using machine learning approaches.

One of the most recent state-of-the-art machine learning approaches for automatic rigging is RigNet [55]. Given an input 3D model representing an articulated character, RigNet predicts a skeleton that matches the animator expectations in joint placement and topology. It also estimates surface skin weights based on the predicted skeleton. One of the main advantages of RigNet is that it can be used for any articulated character not only bipedal human like characters. RigNet is not restricted by a predefined skeletal structure, but on the other hand the user cannot specify the amount of joints or guide their positions, which also means that generated rigs can not follow motion capture data for animation of the mesh if the structure does not match. The RigNet architecture consists of several modules:

- Skeletal joint prediction: A weight function is learned over the input mesh representing the mesh attention. This attention is clustered, and the final joint locations are extracted.
- Skeleton connectivity prediction. This module takes as input the predicted joints from the previous step, including a learned shape and skeleton representation, and outputs a probability representing whether each pair should be connected with a bone or not. The bone probabilities are used as input to a Minimum Spanning Tree algorithm that prioritizes the most likely bones to form a tree-structured skeleton.
- Skinning prediction. Given a predicted skeleton, the last module of our architecture produces a weight vector per mesh vertex indicating the degree of influence it receives from different bones

RigNet also allows optional user input in the form of a single parameter to control the level-of-detail, or granularity, of the output skeleton, however this still does not solve the problem of not being able to specify the amount of joints. Another disadvantage of this approach is that custom meshes should have between 1k to 5k vertices, which can be solved for some meshes with

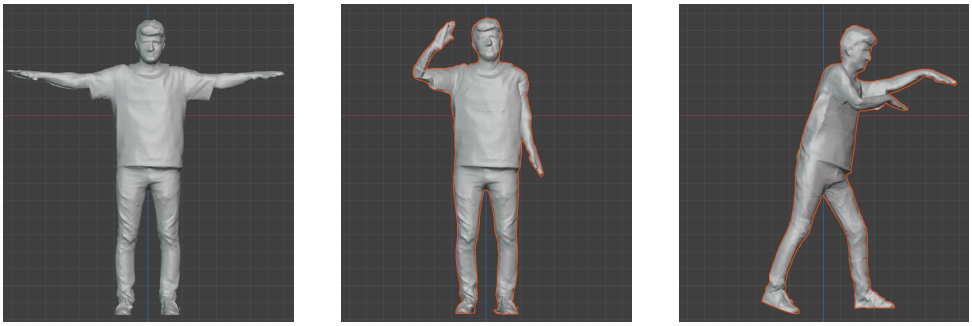
their simplification but for complex meshes reducing the number of vertices will inherently lead to loss of detail and geometry information.

For the disadvantages listed for the above methods the choice of our rigging and skinning method will be Neural Blend Shapes (NBS) [29].

5.1 Neural blend shapes

NBS [29] presents a neural technique that learns rigging, skinning and blend shapes for an input mesh in a T-pose. Using a prescribed skeletal structure makes the generated model compatible with mocap. Aside from rigging and predicting skinning weights the method additionally computes a set of corrective, pose-dependent shapes that improve the deformation quality in joint regions, coined neural blend shapes. An example of taking an input mesh rigging and skinning it with NBS and the following mocap data can be seen in Figure 5.1. The NBS method is trained on the SMPL dataset [32]. This dataset provides a rich set of blend shapes, including ten shape (pose-independent) and 207 pose-dependent shapes. These shapes enable generating a wide variety of human body types (e.g., height, weight, proportions) and high-quality deformations based on joint rotations provided by the SMPL model. However, the SMPL shapes represent relatively anatomically simple characters (nude and hairless). An additional dataset of clothed humans is used from the Multi Garment Network [7]. This dataset focuses on clothed human characters, which is particularly relevant for our goal of rigging clothed human characters. The Multi Garment Network dataset is also used for 'garment augmentation', which extracts garments and adds them to other SMPL models. This approach allows the NBS method to learn how clothing interacts with the underlying body during various poses, leading to more realistic character animation with clothing.

While Pinnochio could also fit the same skeletal structure as neural blend shapes, the skinning can contain notorious Linear Blend Skinning [34] (LBS) artifacts. This method computes the deformation of the mesh as a weighted sum of the character's bone transformations. LBS is a popular skinning method due to its speed and simplicity and due to that it can be easily parallelized to fully utilize modern GPUs' high performance, making the method an essential technique for real-time applications, such as games. For the same reasons Dual Quaternion Skinning [26] (DQS) is a popular choice. These methods require as input the skinning weights per vertex which are either interactively painted and edited [4], or automatically estimated based



(a) : Input mesh. **(b) :** Wave mocap **(c) :** Dance mocap

Figure 5.1: Example of an input human mesh in T-pose 5.1a following mocap data of waving 5.1b and dancing 5.1c respectively. The input mesh was obtained with the neus-facto method.

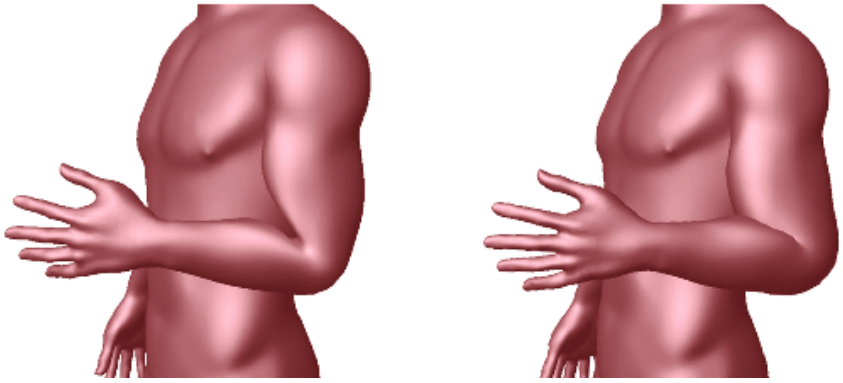


Figure 5.2: Illustration of artifacts produced by LBS (left) and Dual Quaternion Skinning (right). LBS has loss of volume in the elbow joint, DQS has a joint-bulging artifact. Image taken from [42].

on hand-engineered functions of shape geometry and skeleton [4] [5].

It is difficult for such geometric approaches to account for any anatomic considerations implicit in input meshes, such as the disparity between animator and geometric spines, or the skin flexibility/rigidity of different articulations. An example of these artifacts for a human mesh can be seen in Figure 5.2. NBS tries to overcome these issues by learning neural blend shapes, a set of corrective pose-dependent shapes which improve the deformation quality in the joint regions.

The NBS framework addresses the challenge of animating characters with potentially different underlying deformation models used during training. The framework consists of two main branches:

- Envelope Deformation Branch: this branch focuses on learning pose-invariant parameters that define the character's rigging and skinning.

These parameters essentially create a "skeleton" that controls the overall articulation of the character. The learned skeleton is then bound to the character's initial geometry using estimated skinning weights. Combining these weights with joint rotations allows for pose-based articulation of the character's shape.

- Residual Deformation Branch: this branch learns pose-dependent residual displacements that refine the overall character shape based on specific poses.

The framework outputs three main components.

- Rigging: this defines the underlying skeletal structure that controls the character's articulation. The number of degrees of freedom and the hierarchy of the skeleton are pre-defined within the network architecture.
- Skinning: this defines how the skeleton influences the deformation of the character's mesh. Skinning weights determine how much each bone in the skeleton affects specific vertices on the mesh.
- Blend Shapes: Inspired by SMPL [32], these are neural network-generated representations of shape variations used to further refine the character's pose. Unlike SMPL, however, the NBS approach allows for a variable number of blend shapes to be learned.

The network is trained without directly accessing the ground truth rigging, skinning, and blend shapes of the training characters, instead, it leverages indirect supervision. The network observes the relationship between provided joint rotations and the resulting articulated vertex positions. By analyzing this relationship, the network learns to represent the articulation of each character using the pre-defined envelope model, even if the training data utilized is different from the underlying deformation models.
The network can handle characters with potentially different underlying deformation models used during training. The network automatically generates a smaller set of compact and localized blend shapes that are inherently pose-dependent. This eliminates the need for pre-defined, pose-specific blend shapes as training data.

This framework offers an approach to rigging and animating characters that adapts to variations in underlying deformation models without requiring explicit supervision for rigging, skinning, or blend shapes.

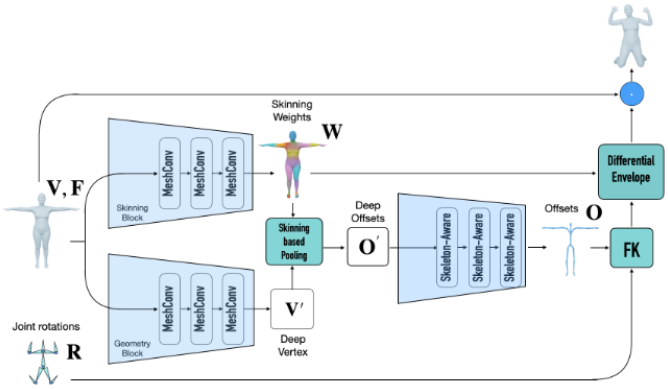


Figure 5.3: The envelope deformation branch overview. Image taken from [29].

5.1.1 Envelope deformation branch

This section dives into the details of the envelope deformation branch within the network architecture. This branch focuses on predicting two key components for character animation: rigging and skinning.
The network receives a triangle mesh as input. This mesh is represented by its vertices $V \in \mathbb{R}^{V\times 3}$, where V is the number of vertices and each vertex has 3D coordinates. Additionally, the mesh's faces (triangles) are defined by the set F.
The outputs of this method are skeletal offsets $O \in \mathbb{R}^{J\times 3}$ and a skinning weight matrix $W \in \mathbb{R}^{V\times J}$. The network predicts offsets O for each joint relative to its parent joint within a predefined skeletal hierarchy. This hierarchy defines the character's skeletal structure and has J total joints. The skinning matrix W defines how the skeleton influences the mesh deformation during animation. Each entry in the matrix represents the weight of a specific joint in affecting a particular vertex on the mesh.
The overview of the envelope deformation branch can be seen in Figure 5.3.

The skinning parameters are obtained with a series of three mesh convolution blocks using the MeshCNN [18] operators. Analogously to classic CNNs, MeshCNN combines specialized convolution and pooling layers that operate on the mesh edges of triangular meshes. Convolutions are applied on edges and the four edges of their incident triangles (in a manifold mesh an edge belongs to two faces/triangles, except the shared edge four additional edges are part of the two triangles), and pooling is applied via an edge collapse operation that retains surface topology, thereby, generating new mesh connectivity for the subsequent convolutions. MeshCNN assumes it operates on a manifold mesh so that each edge is incident to two faces (triangles) at most, and is therefore adjacent to either two or four other edges. The input edge feature is a 5-dimensional vector for every edge: the dihedral angle, two inner

angles and two edge-length ratios for each face. The edge ratio is between the length of the edge and the perpendicular line for each adjacent face. However, the input features used by the NBS methods differ. For each edge the average positions of its two adjacent vertices are calculated. Furthermore, one out of five of the output channels in each hidden layer are max pooled, then the procedure is repeated, and the result is concatenated along the edge axis to extend the receptive field. After a forward pass, in order to predict per-vertex values, the adjacent edge features of the corresponding vertex based on the mesh connectivity are averaged (similar to Point2Mesh [19]) to get the skin matrix W.

The rigging parameters are $O \in \mathbb{R}^{J\times 3}$ of a specific skeleton hierarchy that consists of J offsets. These parameters are learned from a triangular mesh in T-pose as input. Intuitively, each offset O_j of the character's rig can be inferred from its surrounding mesh vertices. To learn a vertex representation that fits that task, we first pass the edge representation of V (similar to the skinning block) through several MeshCNN blocks to obtain a learned deep vertex representation $V' \in \mathbb{R}^{V\times K}$ with K channels. Then, the output skinning matrix is used to apply a skinning based pooling on the deep vertices, which collapses the V features into a set of J deep offsets using the relative skinning weight via

$$O_j^{'} = \frac{\sum_{i=1}^{V} W_{ij} V_i^{'}}{\sum_{i=1}^{V} W_{ij}} \tag{5.1}$$

where $O_j^{'} \in \mathbb{R}^K$ represents a deep feature corresponding to the j-th offset, and W_{ij} is the skin weight that ties vertex i to offset j. This operation is similar to attention based pooling, and ensures that each offset is calculated only as a function of the vertices that are bound to it.
With the predicted deep offsets $O^{'} \in \mathbb{R}^{J\times K}$ the explicit skeleton offset to predict the rig can be estimated. The explicit offset $O \in \mathbb{R}^{J\times 3}$ is predicted using a block of skeletal aware operators [3]. These operators are skeleton-aware, meaning that they explicitly account for the skeleton's hierarchical structure and joint adjacency, and together they serve to transform the original motion into a collection of deep temporal features associated with the joints of the primal skeleton. A primal skeleton is a common skeleton for different skeletons after reducing them with a sequence of edge merging operators. Retargeting can be achieved simply by encoding to, and decoding from this latent space. Since the skeletal topology is fixed in the network, we can exploit joint connectivity, such that each offset is calculated only by its corresponding deep offset and its close neighbors.

During each training iteration, a random pose is injected into the network. This pose is represented by a set of local joint rotations R_i, where each R_i is a 3x3 rotation matrix. These joint rotations guide the deformation of the input character mesh based on the network's predicted rigging and skinning

parameters. The network performs a two-step conversion process to translate the local joint rotations and offsets into a format suitable for deforming the mesh.
Forward Kinematics: Local rotations and offsets are accumulated along the kinematic chain (starting from the root joint) for each joint. This step essentially calculates the global transformation for each joint based on its local rotation and the relative positions of its parent joints.
A differential LBS layer calculates a global transformation for each vertex based on the predicted skinning weight matrix W and the global joint transformations $T_i \in \mathbb{R}^{4\times4}$.

$$T_{R_j} = \sum_i W_{ji} T_i \tag{5.2}$$

Once the per-vertex global transformations T_{R_j} are calculated, they are applied to the input mesh vertices V using a per-vertex operation.

$$\hat{V}_R = T_R \odot V \tag{5.3}$$

This operation essentially deforms the mesh based on the predicted rigging and skinning and the injected pose. An l^2 loss function is used to measure the difference between the predicted deformed mesh $\hat{V}_R$ and the ground truth mesh in the target articulated pose V_R. This loss guides the network to refine its predictions for rigging and skinning parameters during training.

$$\mathcal{L}_V = \|\hat{V}_R - V_R\|^2 \tag{5.4}$$

Overall, the training process iteratively injects random poses, converts local rotations and offsets to global transformations, deforms the mesh, and compares the predicted deformation with the ground truth to guide the network in learning accurate rigging and skinning parameters.

5.1.2 Residual deformation branch

This section introduces the residual deformation branch, which builds upon the concept of blend shapes to further enhance the character's deformation quality during animation. Blend shapes are pre-defined variations of a character's mesh used to capture subtle details and improve animation realism. This method's approach leverages a neural network to predict a set of fixed residual shapes that are combined with pose-dependent coefficients to refine the character's deformation.

The branch takes the input character's vertex positions V and connectivity information F as input. It utilizes pre-trained skinning and geometry blocks (with fixed weights) obtained from the envelope deformation branch.
The output skinning weight matrix W from the pre-trained blocks is combined

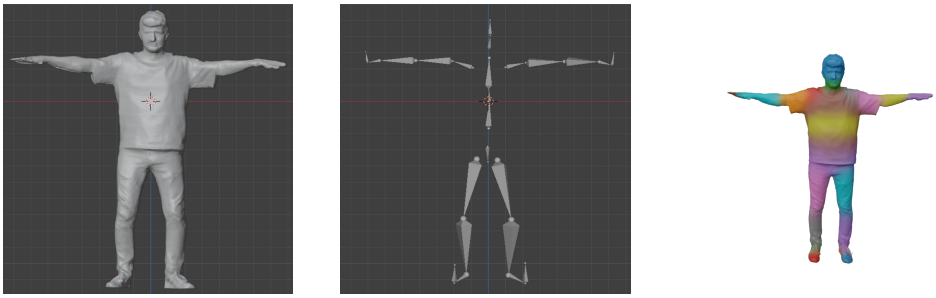


Figure 5.4: Result of NBS on a custom mesh producing a T-pose mesh with associated skeleton and skinning weights.

with the deep vertex representation $V^{'}$ along the channel dimension. This creates a richer feature representation ($V^{'}, W \in \mathbb{R}^{V\times(K+J)}$) that incorporates both vertex information and their relationship to the skeleton. This information is crucial for generating effective blend shapes.
Similar to the envelope branch, mesh convolutions are applied to the edge feature representations of the combined features (V', W). This process results in a set of N residual shapes denoted as B_i where i ranges from 1 to N, and each B_i is a matrix of size $\mathbb{R}^{V\times3}$ representing a specific residual shape.
In parallel to the residual shape generation, a separate small neural network predicts pose-dependent coefficients. This network consists of J MLP blocks (one for each joint) that each process a single joint rotation. The output of this network is a set of coefficients α_{ij} for each residual shape i and each joint j. These coefficients determine the contribution of each residual shape to the final deformation based on the pose.
The final deformed mesh $\hat{V}$ is computed by:

$$\hat{V} = V + \sum_{j=1}^{J}\sum_{i=1}^{N} \alpha_{ij} m_j B_i \tag{5.5}$$

Where m_j is a binary mask to each joint. This mask ensures that only vertices associated with that specific joint (based on the skinning matrix) are influenced by the corresponding residual shapes. This enforces localization and prevents unwanted deformations in areas controlled by static joints. The loss function, similar to the envelope branch, measures the difference between the predicted deformed mesh and the ground truth mesh in the target articulated pose 5.4. This guides the network to refine its predictions for residual shapes and pose-dependent coefficients during training.

The example outputs of NBS on a custom mesh can be seen in Figure 5.4.

Figure 5.5: Example of NBS failing for a non-manifold mesh (right) and a mesh that has been not aligned (right).

5.1.3 Mesh cleanup and alignment

NBS requires as input a mesh that does not contain non-manifold geometry and is aligned with an example mesh to match its orientation, position and scale. Meshes generated with neus-facto don't guarantee this. For this reason we'll have to preprocess the meshes before they are ready as input for Neural Blend Shapes. Examples of failures due to not satisfying the conditions for a mesh can be seen in Figure 5.5.

Generated meshes can include floating artifacts not connected to the main mesh. This can be due to the fact that estimated masks, camera positions contain some errors. The surface reconstruction methods themselves don't have a requirement to reconstruct a single object. For this reason the potential floaters have to be dealt with. With a simple assumption that the human object is the one with the largest amount of vertices this problem can be trivial. If this assumption is not true then that would imply bigger problems, such as complete failure of the human segmentation part of the pipeline. With this knowledge only a simple script that keeps the largest connected component as the final mesh is necessary.

Good connectivity means that the mesh should not have non-manifold geometry. A non-manifold geometry is a 3D shape that cannot be unfolded into a 2D surface with all its normals pointing the same direction. Non-manifold geometry can be caused by various means, such as self intersection of faces, vertices occupying the same space, inner faces, edges sharing more than two faces, surfaces connected to a single vertex, zero volume shapes. In

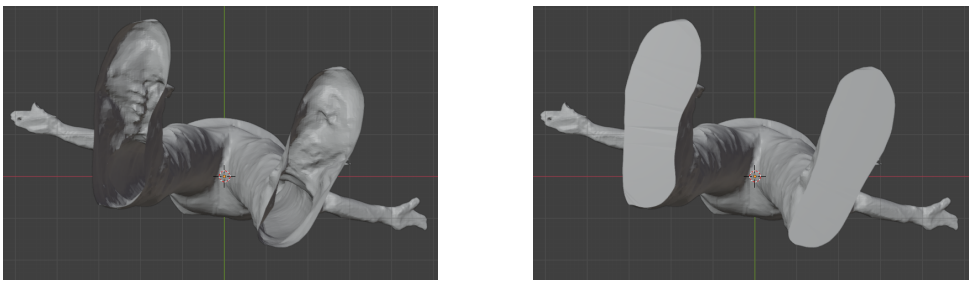


Figure 5.6: Illustration of the cleanup process from the bottom view of the mesh. Zero volume shape (left) being closed to produce a tight solid mesh (right).

this case the most often encountered problem is of the zero volume shape that is caused by holes appearing in the surface of the mesh and since the generated mesh has no thickness it does not have a volume. These holes most often appear in the soles of the feet since in the input video the person is standing still in the T position and the soles of the feet are not directly observable. The proposed fix is creating a script for the Blender [13] a 3D modeling and rendering software. Blender includes helpful functions for our use case such as selecting non-manifold vertices and edges, filling holes and more. A script is created that deletes loose geometry, ..., fills holes in mesh in order to help remove non-manifold geometry. Example result of this cleanup process can be seen in Figure 5.6.

Neural Blend Shapes requires the input mesh to be spatially aligned with a reference mesh. They should be approximately in the same orientation, position and scale. Generated meshes we cannot assume any of these properties to be satisfied by default. This problem is solved by finding the appropriate scale, translation and rotation parameters that minimize the Chamfer distance between the source and target mesh. The parameters are $T \in \mathbb{R}^3$ which is the translation vector, $s \in \mathbb{R}$ the scale factor and $\theta_x, \theta_y, \theta_z \in \mathbb{R}$ which represent the rotation around individual axes. For vertices $V \in \mathbb{R}^{N\times 4}$ where N is the number of vertices, we want to find vertices $V' \in \mathbb{R}^{N\times 4}$ using the same parameters for all vertices which have the minimal Chamfer distance to the reference mesh with vertices $U \in \mathbb{R}^{M\times 4}$ where M is the number of vertices in the reference mesh. V' is obtained by this order of operations:

$$V' = VMR_xR_yR_z \tag{5.6}$$

Where $V = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ . & . & . & . \\ x_N & y_N & z_N & 1 \end{bmatrix}$, $M = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$ where M scales and moves the mesh. $R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x & 0 \\ 0 & -\sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ is the rotation around the

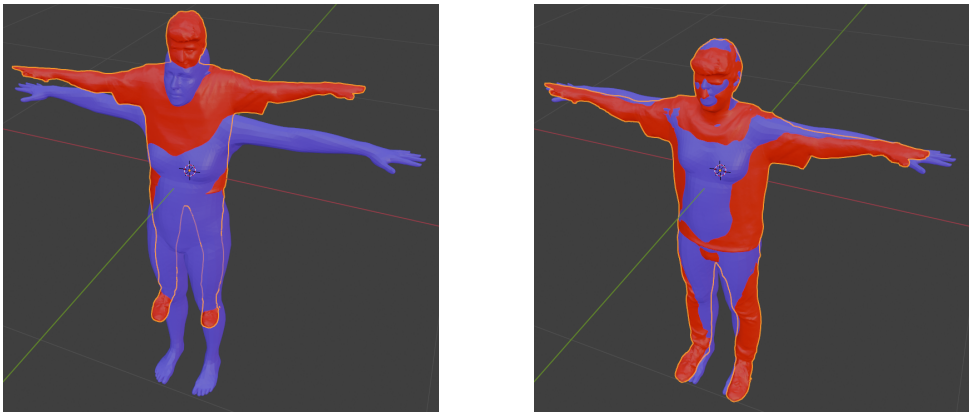


Figure 5.7: Illustration of the alignment process. On the left the reference mesh is in blue and input mesh in red. On the right the input mesh is transformed and aligned to the reference mesh.

x-axis, R_y and R_y are the rotations around their respective axes. Chamfer distance is computed for V' and U see section 3.15. An optimization loop is run minimizing the Chamfer distance based on the parameters using the Adam optimizer [27] with default parameters. Configurable parameters for the scripts are a number of initial guesses for the parameters and number of iterations per guess. An example of this process can be seen in Figure 5.7.

With these preprocessing steps done the mesh is ready to be automatically rigged, skinned and animated with Neural Blend Shapes.

Chapter 6

End to end pipeline

This chapter describes the process and implementation of the end to end pipeline for 3D human reconstruction and automatic rigging presented in this thesis.

The pipeline is implemented as a script that takes as an input a video where the camera moves around a person that stands still in the T-pose position, and outputs a rigged 3D model of that person along with an example animation. The pipeline steps are:

- Pre-processing steps
  - Extracting frames from the video. Since during the training all of the extracted frames are loaded into memory, this could cause memory issues for a longer video. The frames are downsampled keeping every n-th frame where n is a parameter. Images can be downscaled as well.
  - Estimating camera positions. To estimate camera position COLMAP [43], [44] is used. COLMAP is a general-purpose Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipeline with a graphical and command-line interface.
  - Keypoint estimation for the first frame of the video. Ideally the person in the first frame is mostly visible and facing the camera. Keypoint estimation is done using the MMPose library [14].
  - Segmentation with the Segment Anything Model [28]. The person in the first frame is segmented using the keypoints as segmentation prompts for the SAM model.

    - Predicting the rest of the frames in the video. Using the first mask obtained with SAM the rest of the frames are estimated with a semi supervised video object segmentation method XMEM [10].

- Training the neus-facto model to learn the SDF rpresentation of the person in the video taking the frames, estimated camera positions and masks as input.

- Post-processing steps

    - Mesh extraction from the SDF representation of the model using the Marching Cubes [33] algorithm.
    - Mesh cleanup. Deleting all except the largest connected component in the mesh to remove floating artifacts.
    - Filling holes and fixing non-manifold geometry. The mesh extraction doesn't guarantee to produce a manifold mesh. The custom script tries to fill the holes in the mesh, delete loose geometry and etc. to try and fix this problem.
    - Texturing the mesh. Unwrapping the mesh and filling an empty texture image using the color head of the neus-facto model.
    - Mesh alignment. The rigging and skinning process requires the mesh to be aligned with a reference mesh. Meshes are aligned using an optimization loop to find parameters of scale, translation and rotation by minimizing the Chamfer distance between meshes.

- Rigging and skinning. The rigging and skinning parameters are estimated using the Neural Blend Shapes method producing a skeleton and skinning weights from the model.

- Using the rigged model an example animation is produced where the model follows the mocap data.

Extending on the pipeline steps.
For the frame extraction it might be unintuitive how many frames are needed for the training. A good value is 300 frames which SDFStudio [59] uses by default. Downscaling of the images should only really be used when the frames can't fit into memory.
Using COLMAP to estimate camera positions on a custom video is the de facto method for neural reconstruction [49], [50], [30] and novel view synthesis with NeRF [35], [24], [52]. COLMAP can reconstruct the scene as well and the sparse point cloud produced can be used to visualize and set the bounding box and the initial sphere around the subject of interest. The bounding box is to determine from where the points on a ray should be sampled. The sphere is set as the initialization of the SDF field. For our use case a unit bounding box and sphere at the center of SDFStudio's coordinate center have been found to be sufficient and manual alignment of the bounding box is

not necessary. Also using COLMAP puts some limitation on the kinds of videos that are appropriate. Videos taken should avoid texture-less images (e.g., a white wall or empty desk) a more varied background is preferred. The video should not have too much variation in illumination conditions. Keeping to these guidelines might help COLMAP extract features from images and match them more easily, therefore reducing the potential of camera position estimation failing.
The new process of human video segmentation starting with the human keypoint estimation, followed by using the keypoints as segmentation prompts for SAM for the first frame and lastly predicting the rest of the frames with XMEM is described in greater detail in chapter 4.
The training of the neus-facto model can be modified by many parameters. For our use case we use a default configuration that was used on the custom real world datasets. The main configuration parameters are the number of iterations (200k as default), usage of foreground mask (true), number of rays per batch (2048), loss coefficients, learning rates and etc. The default config file can be found in the supplementary material.
The marching cubes resolution parameter is 512. Which defines that the cubes that the algorithm is going to process are $512 \cdot 512 \cdot 512$.
Removing the floating artifacts is simple under the assumption that the largest connected component is the human subject.
Repairing non-manifold geometry for the mesh is not easy. The scripts delete loose geometry and removes doubled vertices, edges with no length, faces with no areas, filling holes in the mesh. All these operations are done with the blender software. A blender add on Blender 3D kit is used as well to help clean the mesh, which is standardly used to help fix non-manifold meshes for 3D printing, as non-manifold geometry is a common problem for 3D printing software.
The texturing process is well described here in section 3.5. The texture quality is dependent on the mesh unwrapping and the color head of the neus-facto model.
The mesh alignment process randomly initializes the scale, rotation and translation parameters and minimizes the loss starting those parameters. By default, the process takes 10 guesses and give 100 iterations to each. The best loss and its parameters are saved and then applied to the source mesh to get the final result.
For the NBS part the chosen mocap data animation is a simple greeting animation where the model raises one hand and waves. The output can either be a sequence of obj files or single fbx file. By default, the fbx output for the animation is used by default.

The script can be launched from any part by specifying a step parameter. This is useful when trying to restart from a specific point in the script with different parameters or modifying the output mesh and running the rest of the script to see a comparison.
Most of the scripts called by the main pipeline and others are Python scripts

using the PyTorch [37], PyTorch3D [41] libraries among others such as numpy, scipy, matplotlib. Additionally, bash scripts are provided when multiple various system commands need to be called such as the end to end pipeline script.
Although using neural SDF representation allows the data to be captured easily with any device which can be used for video recording such as smartphones and rids us of a need for more complex setups of synchronizing multiple cameras for a multi view setup or having access to expensive high quality scanning equipment. There are still some hardware requirements. The training can be especially time-consuming and memory intensive depending on the amount of frames used and their resolution. SDFstudio utilizes libraries that depend on the CUDA framework. For these reasons a machine equipped with a higher end Nvidia GPU is required.
The complete pipeline and experiments were tested on machine containing a GPU NVIDIA GeForce GTX 1080 with 12 GB of RAM and CPU 4x Intel(R) Core(TM) i7-10700KF and another machine equipped with a GPU NVIDIA GeForce RTX A600 with 51.5 GB of RAM and CPU 4x Intel(R) Core(TM) i7-10700KF.
The complete pipeline script can be found in the attachments as **demo.sh** as well as the other scripts for the pipeline and experiments on the datasets. Additionally, README files in the attached materials explain the minimal environment setup as well as how to prepare the datasets for the experiments. Lastly, an implementation of a simple web application is provided for an easier user interaction.

The custom dataset is available for download at `https://drive.google.com/file/d/1awDaKDnSu0qn6C43eUsNPk0XFUpLYqi8/view?usp=sharing` and the outputs of the pipeline on the custom dataset are available for download at `https://drive.google.com/file/d/1leXTQgjrBjYtDzsKYiA4FLD3YArQr9Wh/view?usp=sharing`

6.1 Web application

In this section a simple web application for easier user interaction is presented. The application consists of the server and client part. The Flask web server is hosted on a machine that can handle the training process. The web part is accessible on the local network.

The web application consists of a login page and a home page. The users can login with username and password, these can be set when starting the web server. After the login the user is redirected to the home page. On

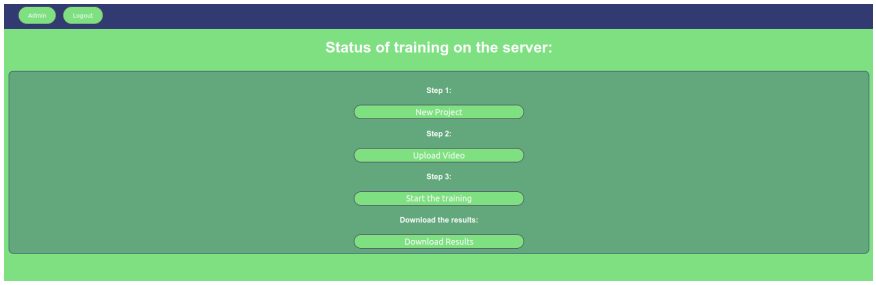


Figure 6.1: Home page of the webapp that can start the end2end pipeline.

the home page the user can create a new project and upload a video to be processed. The server creates a folder identical with the project name and saves the video there. Then the user can start the training which spawns a thread that runs the end to end pipeline script on the uploaded video. User can check on status of the process and get either of NONE when no training was started, ERROR if the training finished in error, FINISHED if the training finished successfully and results are to be downloaded and IN_PROGRESS when the end to end pipeline script is still in execution. Finally, the user can download the results which is a zip file containing the rigged human mesh with its skeleton, texture and example animation. The home page can be seen in Figure 6.1.

Chapter 7

Conclusions

This thesis successfully developed an end-to-end pipeline for reconstructing and rigging 3D clothed human characters from monocular videos. The pipeline takes a single video of a person in a predefined pose as input and outputs a rigged mesh capable of following motion capture data, along with an example animation.

A custom process of human video segmentation was introduced to improve the 3D reconstruction step by separating the person from the scene. The custom process consists of estimating human pose keypoints from the first frame of the video and using those as an input to a promptable segmentation model. The mask obtained for the first frame is used as an input to semi supervised video object segmentation method, which produces masks for the rest of the frames. Evaluation of this approach was done on the TikTok dataset and compared with state-of-the-art methods for frame-wise segmentation. The proposed approach achieved high performance and was ultimately chosen as the segmentation method for this thesis's approach.
Machine learning methods for 3D reconstruction were evaluated on their capabilities on a human dataset. Specifically neural SDF approaches evolved from NeRF such as Neus, Neuralangelo and neus-facto. Evaluation of the mesh reconstruction quality was done on the RenderPeople dataset where renders of the high quality mesh scans were used as input to the methods and then the mesh quality was evaluated using the Chamfer distance.
To evaluate their novel-view synthesis capabilities the models were evaluated on a custom dataset. The created dataset consists of five videos that have a person standing still while the camera rotates around them. Using the PSNR and SSIM metrics the models were evaluated. However, the meshes extracted from these trainings contain visually notable errors, mainly pro-

trusions around the hands and head. Holding still in a T-pose position for a prolonged time is unnatural and forces a drop of the arms over time.
A rigging and skinning process is added to the pipeline. Using Neural Blend Shapes (NBS) on the meshes generated on the custom dataset, the meshes were animated and rigged. Using a predefined skeleton the meshes are able to follow motion capture data creating animations. Custom pre-processing steps of mesh cleanup and alignment were implemented to prepare the reconstructed mesh for the rigging process.
And end-to-end pipeline was successfully implemented alongside a web application for easier user interaction. However, we notice that this approach is highly sensitive to the quality of the input video, mainly in the subject being able to stand still in the predefined position. Due to this sensitivity other 3D human reconstruction approaches from a video might be more suitable, which even allow for the subject to move in the video, however this approach still has benefits such as the resulting output is already in a format readily usable by many 3D software applications and putting minimal constraints on the shape of the human body other than being able to stand in the T-pose.

Potential improvements and further work. Modifying the NBS process so that it can take a model in a more natural pose than the T-pose, such as the A-pose. Or using an alternative rigging process to alleviate this issue.
Evaluation of using additional inputs for the reconstruction method, such as depth maps or using a reference human mesh for the SDF field initialization.
Adding a more detailed template skeleton to allow the model to follow and perform more complex animations, since the current skeleton does not contain any detail for the fingers or face.

Appendices

Appendix A

Bibliography

[1] Renderpeople. https://renderpeople.com/. Accessed: 2024-03-01.

[2] xatlas. https://github.com/mworchel/xatlas-python. Accessed: 2024-05-01.

[3] K. Aberman, P. Li, D. Lischinski, O. Sorkine-Hornung, D. Cohen-Or, and B. Chen. Skeleton-aware networks for deep motion retargeting. *ACM Transactions on Graphics (TOG)*, 39(4):62–1, 2020.

[4] S. Bang and S.-H. Lee. Spline interface for intuitive skinning weight editing. *ACM Transactions on Graphics (TOG)*, 37(5):1–14, 2018.

[5] I. Baran and J. Popović. Automatic rigging and animation of 3d characters. *ACM Transactions on graphics (TOG)*, 26(3):72–es, 2007.

[6] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

[7] B. L. Bhatnagar, G. Tiwari, C. Theobalt, and G. Pons-Moll. Multi-garment net: Learning to dress 3d people from images. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5420–5430, 2019.

[8] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[9] B. Cheng, B. Xiao, J. Wang, H. Shi, T. S. Huang, and L. Zhang. Higherhrnet: Scale-aware representation learning for bottom-up human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5386–5395, 2020.

[10] H. K. Cheng and A. G. Schwing. Xmem: Long-term video object segmentation with an atkinson-shiffrin memory model. In *European Conference on Computer Vision*, pages 640–658. Springer, 2022.

[11] H. K. Cheng, Y.-W. Tai, and C.-K. Tang. Rethinking space-time networks with improved memory coverage for efficient video object segmentation. *Advances in Neural Information Processing Systems*, 34:11781–11794, 2021.

[12] M. V. M. Cirne and H. Pedrini. Marching cubes technique for volumetric visualization accelerated with graphics processing units. *Journal of the Brazilian Computer Society*, 19:223–233, 2013.

[13] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[14] M. Contributors. Openmmlab pose estimation toolbox and benchmark. `https://github.com/open-mmlab/mmpose`, 2020.

[15] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.

[16] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020.

[17] C. Guo, T. Jiang, X. Chen, J. Song, and O. Hilliges. Vid2avatar: 3d avatar reconstruction from videos in the wild via self-supervised scene decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12858–12868, 2023.

[18] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (ToG)*, 38(4):1–12, 2019.

[19] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or. Point2mesh: A self-prior for deformable meshes. *arXiv preprint arXiv:2005.11084*, 2020.

[20] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[21] Y. Jafarian and H. S. Park. Learning high fidelity depths of dressed humans by watching social media dance videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12753–12762, 2021.

[22] B. Jiang, Y. Hong, H. Bao, and J. Zhang. Selfrecon: Self reconstruction your digital avatar from monocular video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5605–5615, 2022.

[23] T. Jiang, X. Chen, J. Song, and O. Hilliges. Instantavatar: Learning avatars from monocular video in 60 seconds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16922–16932, 2023.

[24] W. Jiang, K. M. Yi, G. Samei, O. Tuzel, and A. Ranjan. Neuman: Neural human radiance field from a single video. In *European Conference on Computer Vision*, pages 402–418. Springer, 2022.

[25] Z. Jin, B. Liu, Q. Chu, and N. Yu. Isnet: Integrate image-level and semantic-level context for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7189–7198, 2021.

[26] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46, 2007.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[29] P. Li, K. Aberman, R. Hanocka, L. Liu, O. Sorkine-Hornung, and B. Chen. Learning skeletal articulations with neural blend shapes. *ACM Transactions on Graphics (TOG)*, 40(4):1–15, 2021.

[30] Z. Li, T. Müller, A. Evans, R. H. Taylor, M. Unberath, M.-Y. Liu, and C.-H. Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8456–8465, 2023.

[31] L. Liu, M. Habermann, V. Rudnev, K. Sarkar, J. Gu, and C. Theobalt. Neural actor: Neural free-view synthesis of human actors with pose control. *ACM transactions on graphics (TOG)*, 40(6):1–16, 2021.

[32] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. Smpl: A skinned multi-person linear model. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 851–866. 2023.

[33] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*, pages 347–353. 1998.

[34] T. Magnenat, R. Laperrière, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface'88*, pages 26–33. Canadian Inf. Process. Soc, 1988.

[35] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[36] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022.

[37] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[38] E. Pittini. Multi-view 3d reconstruction using nerf-based approaches. Master's thesis, University of Bologna, 2023.

[39] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. R. Zaiane, and M. Jagersand. U2-net: Going deeper with nested u-structure for salient object detection. *Pattern recognition*, 106:107404, 2020.

[40] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[41] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020.

[42] N. A. Rumman and M. Fratarcangeli. Skin deformation methods for interactive character animation. In *Computer Vision, Imaging and Computer Graphics Theory and Applications: 11th International Joint Conference, VISIGRAPP 2016, Rome, Italy, February 27–29, 2016, Revised Selected Papers 11*, pages 153–174. Springer, 2017.

[43] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[44] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[45] C. Song, J. Wei, R. Li, F. Liu, and G. Lin. 3d pose transfer with correspondence learning and mesh refinement. *Advances in Neural Information Processing Systems*, 34:3108–3120, 2021.

[46] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems*, 33:7537–7547, 2020.

[47] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, J. Kerr, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. McAllister, and A. Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, 2023.

[48] J. Wang, C. Wen, Y. Fu, H. Lin, T. Zou, X. Xue, and Y. Zhang. Neural pose transfer by spatially adaptive instance normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5831–5839, 2020.

[49] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021.

[50] Y. Wang, Q. Han, M. Habermann, K. Daniilidis, C. Theobalt, and L. Liu. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3295–3306, 2023.

[51] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[52] C.-Y. Weng, B. Curless, P. P. Srinivasan, J. T. Barron, and I. Kemelmacher-Shlizerman. Humannerf: Free-viewpoint rendering of moving people from monocular video. In *Proceedings of the IEEE/CVF conference on computer vision and pattern Recognition*, pages 16210–16220, 2022.

[53] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[54] D. Xiang, F. Prada, Z. Cao, K. Guo, C. Wu, J. Hodgins, and T. Bagautdinov. Drivable avatar clothing: Faithful full-body telepresence with dynamic clothing driven by sparse rgb-d input. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–11, 2023.

[55] Z. Xu, Y. Zhou, E. Kalogerakis, C. Landreth, and K. Singh. Rignet: Neural rigging for articulated characters. *arXiv preprint arXiv:2005.00559*, 2020.

[56] K. Yang, K. Qinami, L. Fei-Fei, J. Deng, and O. Russakovsky. Towards fairer datasets: Filtering and balancing the distribution of the people subtree in the imagenet hierarchy. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 547–558, 2020.

[57] R. Yao, G. Lin, S. Xia, J. Zhao, and Y. Zhou. Video object segmentation and tracking: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(4):1–47, 2020.

[58] L. Yariv, J. Gu, Y. Kasten, and Y. Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021.

[59] Z. Yu, A. Chen, B. Antic, S. Peng, A. Bhattacharyya, M. Niemeyer, S. Tang, T. Sattler, and A. Geiger. Sdfstudio: A unified framework for surface reconstruction, 2022.

[60] J. Zhao, T. Wang, M. Yatskar, V. Ordonez, and K.-W. Chang. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. *arXiv preprint arXiv:1707.09457*, 2017.

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Otgonsuren Rico David** Personal ID number: **481890**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master’s thesis details

Master’s thesis title in English:

3D human model reconstruction and automatic rigging from a monocular video

Master’s thesis title in Czech:

3D rekonstrukce člověka a automatické mapování kostry z monokulárního videa

Guidelines:

The objective of this thesis is to generate a high-fidelity 3D animatable clothed human model from a monocular video. While there are ML methods that solve this task directly [1,2,3], the issue is that these animatable avatars do not produce high quality meshes compared to the ML state of the art methods for general static scenes [4, 5].
The first step consists of exploring the SOTA methods for general-purpose 3d reconstruction.
The second step consists in applying, tuning and comparing some of these methods for generating human meshes.
The third step consists in exploring and applying rigging and animation methods on the generated meshes [6,7,8].
Finally, a complete end-to-end pipeline for generating a rigged human mesh from a video is implemented.
For training of the models two remote machines will be available with GPUs: NVIDIA GeForce GTX 1080 Ti, NVIDIA RTX A6000. Custom, publicly available dataset with at least five subjects will be created.

Bibliography / sources:

[1] Jiang, Wei, Kwang Moo Yi, Golnoosh Samei, Oncel Tuzel, and Anurag Ranjan. "Neuman: Neural human radiance field from a single video." In European Conference on Computer Vision, pp. 402-418. Cham: Springer Nature Switzerland, 2022.
[2] Jiang, Tianjian, Xu Chen, Jie Song, and Otmar Hilliges. "Instantavatar: Learning avatars from monocular video in 60 seconds." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 16922-16932. 2023.
[3] Qian, Zhiyin, Shaofei Wang, Marko Mihajlovic, Andreas Geiger, and Siyu Tang. "3DGS-Avatar: Animatable Avatars via Deformable 3D Gaussian Splatting." arXiv preprint arXiv:2312.09228 (2023).
[4] Li, Zhaoshuo, Thomas Müller, Alex Evans, Russell H. Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. "Neuralangelo: High-Fidelity Neural Surface Reconstruction." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8456-8465. 2023.
[5] Wang, Peng, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction." arXiv preprint arXiv:2106.10689 (2021).
[6] Li, Peizhuo, Kfir Aberman, Rana Hanocka, Libin Liu, Olga Sorkine-Hornung, and Baoquan Chen. "Learning skeletal articulations with neural blend shapes." ACM Transactions on Graphics (TOG) 40, no. 4 (2021): 1-15.
[7] Liao, Zhouyingcheng, Jimei Yang, Jun Saito, Gerard Pons-Moll, and Yang Zhou. "Skeleton-free pose transfer for stylized 3D characters." In European Conference on Computer Vision, pp. 640-656. Cham: Springer Nature Switzerland, 2022.
[8] Xu, Zhan, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. "Rignet: Neural rigging for articulated characters." arXiv preprint arXiv:2005.00559 (2020).

Name and workplace of master's thesis supervisor:

Ing. Davide Catellucci DataVision s.r.o

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **21.02.2024** Deadline for master's thesis submission: _____

Assignment valid until: **15.02.2026**

Ing. Davide Catellucci Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

_____. Date of assignment receipt	_____ Student's signature

