Learning to Optimize in the Energy Landscape of Parametric Head Models

Robin Borth, Matthias Nießner
Technical University of Munich
robin.borth@tum.de

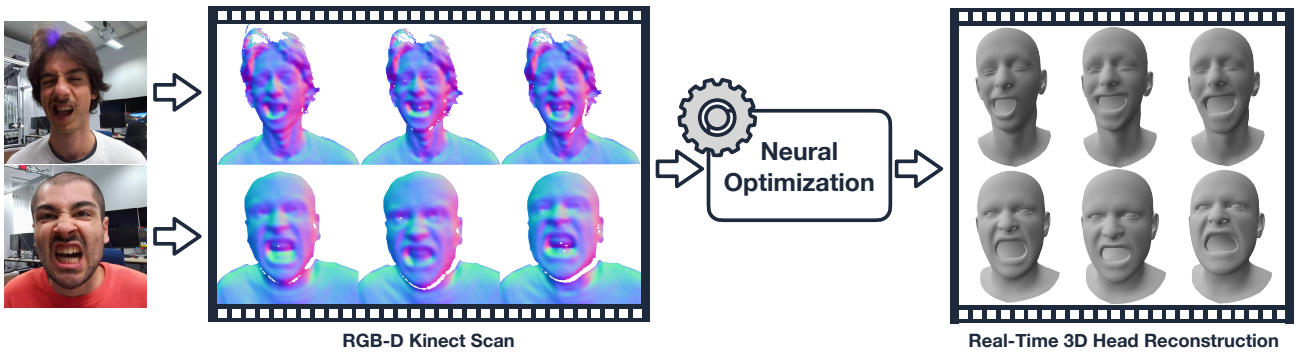

Figure 1. Teaser. We present OptiHead, a novel method for robust real-time tracking of parametric head models. OptiHead takes an RGB-D sequence (left) as input and utilizes a neural optimization pipeline to fit a 3DMM (right) in just a few iterations.

Abstract

We present a novel approach for real-time tracking of parametric head models from monocular RGB-D sequences. To this end, we propose OptiHead, a fully differentiable neural optimization pipeline capable of interpolating between regression and optimization by adjusting the energy formulation used for second-order solvers, enabling real-time tracking in just a few iterations. Building on projective ICP to establish correspondences, we incorporate neural weighting of residual terms using large receptive fields and regularize the optimization process with a learned prior that accounts for the uncertainty of these weights. Moreover, our method can be trained end-to-end, which was previously not done for neural optimization pipelines for 3DMM, which is achieved by utilizing mesh rasterization that is capable of forward-mode automatic differentiation, allowing efficient and differentiable computation of the Jacobian matrix used during Gauss-Newton optimization. Our method outperforms ICP for dynamic facial expressions at 21.29 *FPS, allowing for real-time applications.*

1. Introduction

Real-time tracking and reconstructing of human heads is a fundamental problem across many applications, such as AR/VR, teleconferencing, or virtual avatar animation. In this work, we focus on real-time frame-to-frame monocular RGB-D tracking using a Kinect sensor, a widely adopted setup for practical real-world applications.

A well-established approach to this challenging problem is face reconstruction using 3D morphable models (3DMMs) [2], which leverage PCA-based low-rank approximations of 3D face geometry to represent variations in human faces. These models function as priors to regularize the face reconstruction process, simplifying the tracking task to find the appropriate parameters of a 3DMM.

In the existing literature, two primary approaches for face tracking using 3D morphable models can be identified: optimization-based methods and regression-based methods. Regression techniques, such as those proposed in [8, 9, 28, 30], can predict the parameters of parametric head models from a single image. However, these methods often struggle to capture complex facial expressions accurately and cannot correct prediction errors. Despite this limitation, they enable real-time applications due to the inherent efficiency of feed-forward networks. In contrast, optimization-based approaches [25, 26] enable pixel-accurate registration and are well-suited for frame-to-frame tracking, as they can leverage the state from the previous frame to ensure temporally consistent tracking. However, these methods often require a trade-off between real-time capabilities and reconstruction quality.

We present OptiHead, a fully differentiable neural optimization pipeline capable of interpolating between regres-

sion and optimization by adjusting the energy formulation used for second-order solvers, enabling real-time tracking in just a few optimization iterations. This is achieved through neural weighting of residual terms used during Gauss-Newton optimization and regularization based on a learned prior that accounts for the uncertainty of these weights. Moreover, our method can be trained end-to-end, previously not done for neural optimization pipelines for 3DMM that utilize second-order solvers like Gauss-Newton. This is accomplished by computing the Jacobian matrix in a differentiable manner and solving for the optimization update within an automatic differentiation framework such as PyTorch. More concretely, we propose utilizing forward-mode automatic differentiation (fwAD) based on dual numbers, which exploits the structure of the Jacobian matrix to enable efficient vectorization on the GPU while also tightly integrating a differentiable mesh rasterizer within this framework. Additionally, we utilize standard backpropagation through computational graphs to train the neural components, allowing for efficient end-to-end training and inference speed at a rate of 21.29 FPS. Our contributions are as follows:

- We introduce a novel end-to-end differentiable optimization pipeline based on projective ICP. The pipeline integrates neural weighting and regularization with dynamic correspondence refinement.
- We develop an efficient mesh rasterizer integrated into a differentiable Gauss-Newton optimizer capable of forward-mode automatic differentiation, allowing efficient computation of Jacobian matrices.
- We demonstrate that our neural optimizer can robustly fit highly dynamic expressions in real-time and outperform the ICP baseline.

2. Related Work

2.1. Parametric Head Models

To capture human faces, the seminal work of Blenz and Vetter introduced 3D morphable models [2]. They effectively represent variations of heads using PCA with an underlying 3D mesh representation. More advanced face models were introduced utilizing improved data capture techniques [4, 5], including multilinear models of identity and expression [3], and combining linear blend skinning [17] with articulated head parts [14], which has become the de facto standard for PCA-based methods. Those template-based approaches allow fast-tracking but often lack details, e.g., missing hair. Another approach involves using neural representations [10, 29], which enable detailed reconstructions. However, achieving real-time tracking remains challenging. This paper focuses on achieving real-time tracking by employing the mesh-based FLAME model [14].

2.2. Tracking of Parametric Head Models

Optimization. Prominent methods for face tracking and reconstruction, such as Face2Face [25,26], optimize the parameters of a multi-linear PCA-based model by employing a dense geometric, sparse landmark and regularization loss. Existing approaches achieve real-time facial tracking by leveraging efficient second-order optimizers such as Gauss-Newton (GN) [1] or Levenberg-Marquardt (LM) [19], often solving the linear system using the PCG method [23]. These optimization pipelines typically rely on sparse landmark tracking [18] for robustness. In contrast, our method employs a landmark-free optimization strategy, eliminating the need to detect landmarks altogether.

Regression. Approaches such as DECA [9] directly predict the parameters of a FLAME [14] through regression techniques. The follow-up work EMOCA [9] introduces a novel deep perceptual emotion consistency loss, improving on tracking of dynamic expressions. Furthermore, [28] demonstrates the effective use of transformer-based architectures for monocular 3D face reconstruction.

2.3. Differentiable Optimization

Differentiable optimization based on first-order optimizers [12] typically requires many iterations to converge. However, gradient computation can be performed efficiently [7], as demonstrated by their successful adaption in fitting morphable models to sparse 2D landmarks using neural optimization. Approximations of second-order solvers [1, 19] are more prominent for tracking due to their efficiency. Various differentiable optimization versions exist for non-rigid tracking [6, 11, 15]. To make these methods differentiable, it is typically necessary to compute the Jacobian matrix analytically and use automatic differentiation to propagate gradients. However, analytically deriving the Jacobian for 3DMMs is impractical. In this work, we leverage automatic differentiation to enable fully differentiable optimization with the Gauss-Newton optimizer for 3DMMs.

3. Preliminary

FLAME. A parametric face model serves as our prior and is defined by $\mathcal{F}(\beta, \theta, \psi)$. The vectors β, θ, and ψ control the identity, pose, and expression of the resulting head model and are summarized through x. The output of $\mathcal{F}(x)$ are the vertices of the template mesh $V \in \mathbb{R}^{|V|\times 3}$ in camera space together with face indices $F \in \mathbb{R}^{|F|\times 3|}$ used for rendering.

Mesh Rasterization We employ mesh-rasterization, denoted as $\mathcal{R}$, to compute barycentric coordinates B used to interpolate the vertex attributes V' for rendering the current state $\mathcal{F}(x)$ of the FLAME model,. This process is summarized by the expression $I(\mathcal{R}(\mathcal{F}(x)), V')$. In Section 4.4, we demonstrate how to extend mesh rasterization to enable fwAD, allowing for end-to-end training.

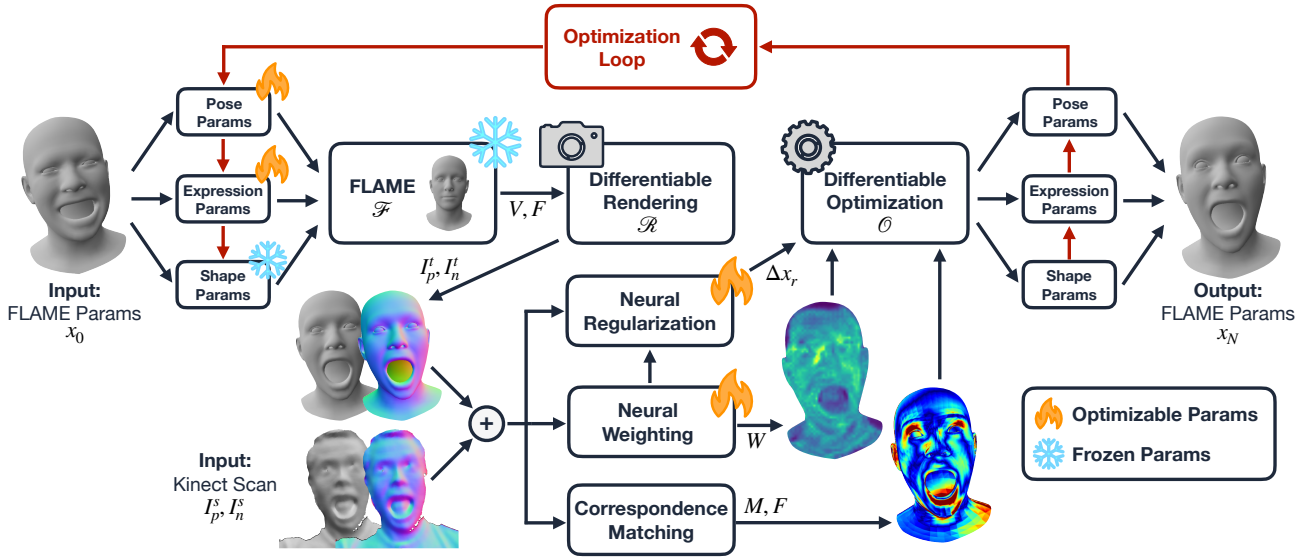

Figure 2. Method Overview. The OptiHead framework is structured around a few end-to-end differentiable neural optimization iterations. At each iteration k, the following steps are executed: (a) the current parameters x_k are input into the flame model $\mathcal{F}$ and rendered as 2D normal I_n^t and point maps I_p^t; (b) these 2D maps are then compared against the source Kinect scan I_p^s, I_n^s to establish correspondences M and residual terms F; (c) the scan and renderings are then concatenated and processed through a neural network to predict weightings W and priors Δx_r used during optimization; (d) a Gauss-Newton optimization step is applied, and (e) this process is repeated for N iterations to obtain the final parameters x_N.

4. Method

Our fully differentiable neural optimization pipeline is based on the projective ICP framework [22] and its extension to 3DMMs [25]. We extend the pipeline by using learnable weighting and regularization terms to guide the optimization more effectively within the energy landscape. During inference, our method operates exclusively within this optimized neural space, see Figure. 2.

4.1. Neural Projective ICP Optimization Pipeline

The parameters x_0^t are initialized from the optimized parameters of the previous frame x_N^{t-1} and passed through the FLAME model $\mathcal{F}$ to obtain the current vertex positions V in camera space together with the face indices F. We then use our differentiable mesh rasterizer R capable of fwAD to render x_0 into 2D normal maps I_n^t and point maps I_p^t. Following projective ICP, we establish pixel-wise correspondence between the Kinect scan I^s and the renderings I^t, resulting in a 2D mask M. To ensure stable training and optimization, we perform outlier removal based on normal t_n and depth t_d thresholds computed between the renderings and the Kinect scan. We then feed I^t and I^s through the neural weighting module to compute a 2D weighting map W and uncertainty estimation feature f_w. This uncertainty estimation is passed through the neural regularization module to compute the weights w_r and offset values Δx_r for each parameter x_k. This information is used to define the residuals for the optimization. The differentiable Gauss-Newton optimizer solves for a delta vector Δx_k and updates the current set of parameters. This can be done in an optimization loop for N steps to get the final parameters x_N, which are then used to define the loss function and train the neural weighting and regularization modules.

4.2. Neural Residual Weighting

We employ the U-Net [21] architecture to predict the weights W. We first concatenate the normal I_n^s and point maps I_p^s from the Kinect scan with the normal I_n^t and point maps I_p^t from the renderings, resulting in an image $I \in \mathbb{R}^{W \times H \times 12}$ containing all information about the current state x_k. We feed I through the encoder $\mathcal{E}$ of the U-Net to obtain the latent code f_w, which is used as input for the neural regularization module. We then apply standard upscaling and skip connections in the decoder $\mathcal{D}$ to predict the weighting map $W \in \mathbb{R}^{W \times H}$.

4.3. Neural Regularization

To interpolate between optimization and regression, we utilize a neural regularization module to predict a weight w_r and neural offset delta Δx_r for the current set of parameters x_k. The module takes as input the parameters x_k alongside the latent representation f_w from the U-Net, which encodes the uncertainty of the current residual weights and correspondences. We use a shared MLP to predict an intermedi-

ate latent f_r, which is further refined by small MLP heads for each parameter to predict both the weights w_r and delta vectors Δx_k. The delta vectors Δx_k are initialized to zero during training. The neural prior x_r is obtained by adding Δx_k back to the current parameters x_k.

4.4. Differentiable Optimization

To make the Gauss-Newton [1] algorithm differentiable, we propose to compute the Jacobian matrix efficiently using forward-mode auto-differentiation based on dual numbers, which is significantly more efficient than reverse-mode auto-differentiation (revAD), see Section 5.5. The network training is performed using standard backward-mode auto-differentiation based on computational graphs.

Differentiable Gauss Newton. The Gauss-Newton optimizer can be described by solving the following equation.

$$J_F(x_k)^T J_F(x_k) \Delta x_k = -J_F(x_k)^T F(x_k) \tag{1}$$

Where $J_F(x_k)$ denotes the Jacobian matrix, while $F(x_k)$ contains the residuals. We propose using an auto-differentiable framework to compute the Jacobian matrix. Specifically, we utilize PyTorch [20] with forward-mode automatic differentiation and composable function transforms to compute $J_F(x_k)$, which tracks the computational graph during the matrix computation. The linear system is solved using PyTorch LU-decomposition for matrices, which has a closed-form gradient computation and is differentiable. The parameters are then updated with:

$$x_{k+1} = x_k - \alpha \Delta x_k \tag{2}$$

where α defines the step size of the optimization.

Differentiable Mesh Rasterizer To support efficient fwAD during mesh rasterization, we propose to pre-compute the barycentric coordinates B outside the composable function transform, allowing the utilization of the graphics pipeline and interpolate the vertex attributes V' inside the composable function transform using PyTorch. This ensures that the gradients can flow through the Jacobian computation during the hole optimization process. This simple modification enables the use of fwAD for Jacobian computation within an analysis-trough-synthesis approach, which is not feasible with other mesh rasterizers, such as SoftRasterizer [16] and NvDiffRast [13], designed for general purposes. The gradient flow can be seen in Figure 3.

Optimization Energy Terms During optimization, our method operates within the energy landscape modified by the neural components. We adopt a neural version of the point-to-plane energy term, defined as:

$$E_{p2p} = \sum_{i \in \mathcal{R}} M^i [W^i * ((p_s^i - p_t^i) * n_t^i)])^2 \tag{3}$$

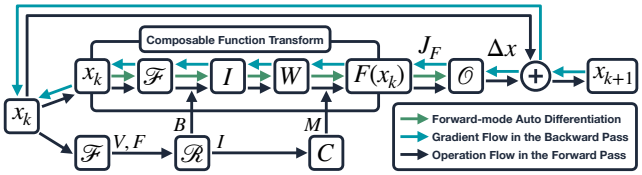

Figure 3. Gradient Flow. The differentiable mesh rasterizer pre-computes barycentric coordinates outside the composable function transform in OpenGL and interpolates the vertex attributes in PyTorch inside the function transform, allowing to compute the jacobian matrix $\mathcal{J}$ using efficient forward-mode auto differentiation.

where we use the normals from the rendering to compute the distance and filter out the residuals based on the correspondence mask M. We replace the zero-latent regularization used in [25] with a neural regularization based on the uncertainty of the weights and correspondences to constraint the optimization more efficiently:

$$E_{reg} = \|w_r(x_k - x_r)\|_2^2$$

Generalized Optimization and Regression The formulation we proposed in the differentiable optimization section can be described as a generalization of optimization and regression techniques. The method can interpolate between both of them depending on the current state x_k. We can show this by reformulating the Jacobian and residuals based on the neural point-to-point and regularization as follows:

$$(J_p^T J_p + \text{diag}(w_r^2))\Delta x_k = -(J_p^T F_p + w_r^2 \Delta x_r^k) \tag{4}$$

The neural optimizer can learn to set w_r^2 to zero, eliminating the neural regression component and resulting in pure optimization based on the point-to-plane term.

$$(J_p^T J_p + \mathbf{0})\Delta x_k = -(J_p^T F_p + \mathbf{0}) \tag{5}$$

To have a regression network, the method could learn to predict zero-weighting maps $W = 0$, which would lead to the formulation:

$$(\mathbf{0} + \text{diag}(w_r^2))\Delta x_k = -(\mathbf{0} + w_r^2 \Delta x_r^k) \tag{6}$$

$$\Delta x_k = -\Delta x_r^k \tag{7}$$

The neural regularizer only needs to learn the delta Δx_r^k between the current and optimal parameters, which can then be solved in a single step, effectively performing regression. Another common approach to regularize 3DMMs is to center them around zero. This can be achieved if the network predicts the delta as $\Delta x_r^k = -x^k$, resulting in zero-latent regularization. This joint optimization formulation, utilizing neural weighting and regularization, allows for flexible interpolation between optimization and regression. This improves convergence speed and enhances robustness. For instance, in projective ICP, with imperfect correspondences during initial iterations, the method can first prioritize the prior and then shift focus to the optimization.

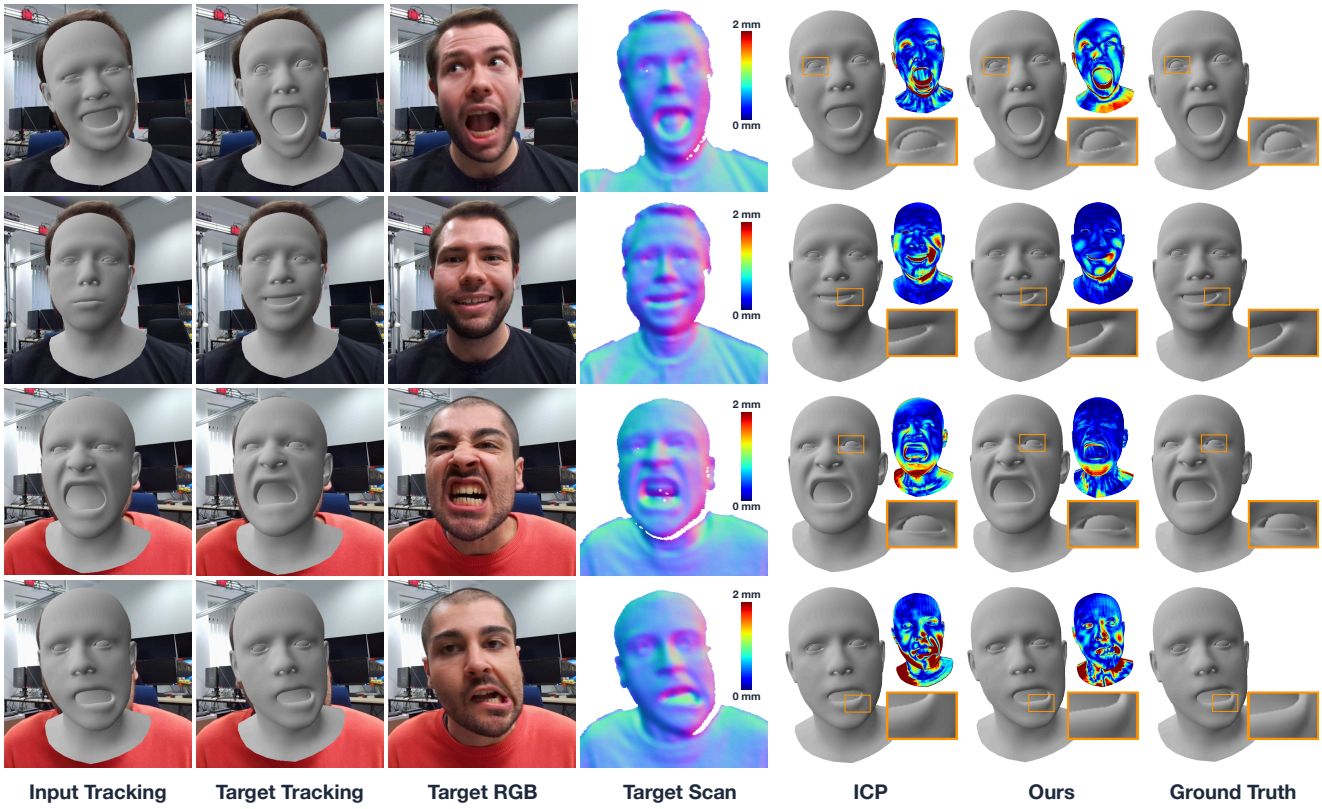

Figure 4. Qualitative Results. Tracking results of the proposed method. On the left, the initial optimization parameters are rendered and overlaid onto the source RGB frame. Next is the target tracking, with the corresponding target RGB frame and Kinect scan shown side by side. On the right, the final real-time results for both ICP and our method are presented alongside the ground truth. The ground truth is obtained using the optimization pipeline from the dataset generation process.

4.5. End-To-End Differentiable Training

Our method enables end-to-end training, as described in the previous sections. To train the neural modules, we propose to utilize a sophisticated optimization pipeline capable of pixel-accurate registrations to gather the ground truth data, which are the parameters x_{gt} of a tracked FLAME model. During training, we perform only a few optimization iterations $N = 2$, forcing the optimization to converge within a limited number of steps. We apply an $L1$-Loss between the ground truth parameters x_{gt} and the final tracked parameters after N steps.

$$L_{param} = \lambda_{param} \|x_N - x_{gt}\|_1 \tag{8}$$

We also apply a geometric loss based on the vertices obtained from an additional evaluation of the FLAME model $\mathcal{F}$, which emphasizes geometric correctness, as similar reconstructions can arise from different parameters and depend on the optimization process used during tracking.

$$L_{geo} = \lambda_{geo} \|V - V_{gt}\|_2^2 \tag{9}$$

The final loss is the sum of the previous losses, which is used to train the weighting and prior modules.

5. Experiments

We aim to evaluate how well the neural optimizer generalizes to dynamic expressions in a sequential setting. We assume that the shape parameters are known and optimizing only for expression and pose parameters.

Synthtetic Dataset To evaluate and pre-train our method with known relationships between scan data and FLAME parameters, we randomly sample 1,000 shapes from the FLAME latent space, each paired with 100 expressions. We use our mesh rasterizer to render normal and point maps from these known parameters to establish the relationships.

DPHM-Dataset To evaluate dynamic face reconstruction, we use the dataset from DPHM [24], which contains 130 single-view depth scan sequences capturing various facial expressions in motion. The dataset includes 26 subjects, each recorded in 5 sequences with 127 frames per sequence. We excluded three individuals from the training set to evaluate our method on unseen subjects in real-world scenarios.

Baselines We initialize the optimization with the parameters from the previous frame and fit them to the target frame within a limited number of iterations to meet real-time con-

Method	FLAME (norm) ↓				P2P (mm) ↓				Vertices (mm) ↓				Time (ms) ↓
	0→1	0→2	0→4	0→8	0→1	0→2	0→4	0→8	0→1	0→2	0→4	0→8	
ICP	0.495	0.543	0.593	0.634	1.784	1.816	1.894	2.025	**6.182**	**6.212**	**6.558**	**7.815**	**36.01**
Ours	0.315	0.421	**0.520**	**0.566**	1.697	**1.738**	**1.817**	**1.956**	6.314	6.362	6.666	7.833	46.97
Ours w/ synth.	**0.289**	**0.413**	0.523	0.572	**1.696**	1.741	1.829	1.967	6.275	6.331	6.715	8.239	46.51

Table 1. Quantitative Results. Results are obtained on the real-world Kinect dataset in the real-time setting. The frame jumps 0→x indicate the index difference between source and target frames. Ours refers to training on the Kinect dataset, while Ours w/ synth. denotes pretraining on the synthetic dataset followed by fine-tuning on the Kinect dataset.

straints. The baseline follows the standard ICP optimization pipeline with point-to-plane and zero-latent regularization, effectively disabling the neural components.

Metrics To evaluate the quality of our reconstruction, we report the $L1$-FLAME distance, $L2$-Vertices distance between known vertex correspondences in millimeters along with the point-to-point distance obtained in a projective ICP style, with outlier removal based on a $1cm$ threshold.

5.1. Implementation Details

Training We implement our method in PyTorch [20] and use OpenGL [27] for mesh rasterization. The model is based on a U-Net architecture with three upscaling and downscaling layers, starting with 32 features. We train the model for 2000 iterations using the Adam [12] optimizer, with a batch size of 64 and a learning rate of $5e^{-4}$. We train on the DPHM dataset using 1 RTX A4000 GPU with 16GB of VRAM, which takes roughly 16 hours.

Dataset Pre-Processing We perform pre-processing steps to remove the background using a depth threshold and smooth the Kinect scans with bilateral filters to reduce noise. We additionally utilized the detected landmarks from [24] to generate our training dataset.

Tracking We apply $N = 2$ optimization iterations during inference and training. We optimize at a low resolution of 256×256 to enhance performance and stability and set the optimization step size to $\alpha = 0.7$, achieving a good trade-off between convergence speed and robustness.

5.2. Tracking Results

We compare our method against the ICP baseline and report the quantitative and qualitative results in Tab. 1 and Figure 4. Our method achieves real-time reconstruction of highly dynamic expressions in real-world settings, showing minimal visual differences compared to the tracking results from a sophisticated optimization pipeline. The effectiveness of our approach for these challenging expressions can be seen in the FLAME metric, which significantly outperforms the ICP baseline and aligns with the visual comparisons. Additionally, the point-to-point metric indicates that our method fits the observable scan more effectively. However, we observe lower performance in the vertices metric, which may be attributed to the lack of an energy term penalizing sparse correspondences.

Method	FLAME ↓	Vertices (mm)↓	Time (ms) ↓
no optim. (base)	1.106	5.14	–
ICP w/ reg.	0.509	0.921	40.04
ICP w/o reg.	0.610	0.960	33.57
w/ single optim.	0.584	1.342	**22.67**
w/ single corresp.	0.488	0.717	37.58
w/ end-to-end (1)	0.194	0.394	45.74
w/ end-to-end (3)	0.192	0.380	45.76
w/o neural weights	0.417	0.658	43.83
w/o neural prior	0.185	0.374	45.49
Ours	**0.182**	**0.363**	45.61

Table 2. Ablations Studies. Comparison of baselines (top), isolated optimization settings from our approach (middle), and variations of our architecture (bottom). Ablation studies were conducted on the synthetic dataset, see (Sec. 5.3) for detailed descriptions of each experiment.

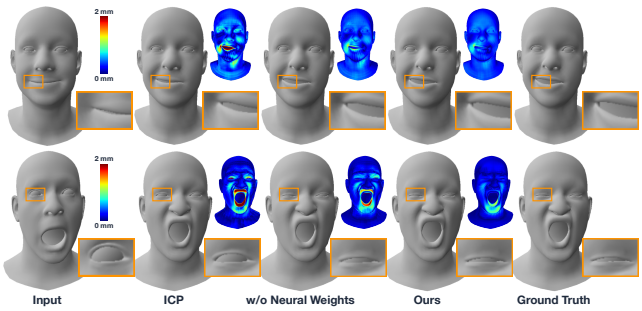

Figure 5. Effect of Neural Optimization. Results on the synthetic dataset show that without the neural weighting and prior module, the model cannot accurately reconstruct the geometry for challenging expressions, e.g., mouth opening and eye closing.

5.3. Tracking Abblations

We support our claims by performing ablations on the synthetic dataset, as reported in Figure 5 and Table 2. The ablation results demonstrate that our design choices improve generalization and enable greater optimization efficiency.

Correspondences The effect of having multiple correspondence updates during the optimization is helpful.

End-To-End Optimization Training the method in an end-to-end fashion helps to discover a more optimal optimization path, leading to improved reconstruction. Here, *w/ end-to-end (n)* refers to a training setup with n iterations, where the evaluation was performed on N iterations.

Effectivenes of Neural Weighting Using an optimized weighting map for the residual terms significantly accelerates convergence and enhances robustness in fewer steps.

Neural Prior Allowing the method to utilize neural regularization and integrate it into the optimization pipeline can improve early-stage optimization. However, this benefit is limited to the synthetic dataset due to the dominant efficiency of neural weighting.

Synthetic Pre-Training Pre-training on the synthetic dataset improves generalization only in the real-world single-frame jump setting, likely due to under-sampling large pose motions compared to the Kinect dataset.

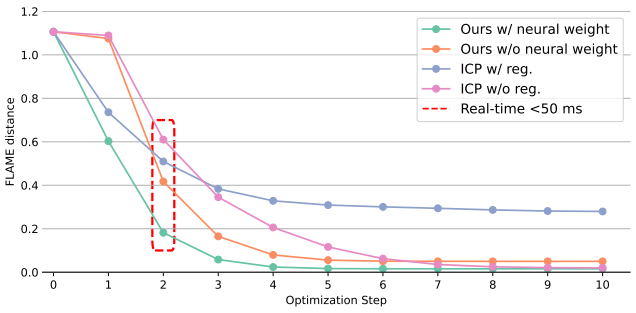

Figure 6. Iteration Performance. Convergence analysis on the synthetic dataset demonstrates that our method outperforms traditional ICP in the few-step setting, enabling accurate real-time tracking of highly dynamic facial expressions.

5.4. Optimization Iterations Effectiveness

Training and evaluation were done with $N = 2$ iterations, which offer a good trade-off between speed and quality, achieving real-time performance at 21.29 FPS. While more iterations still improve the quality, they also increase computation time see Figure 6.

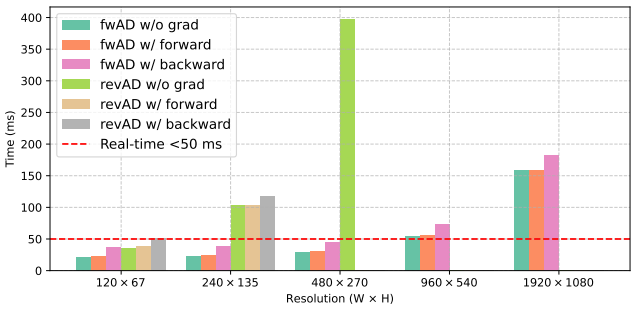

Figure 7. Differentiable Optimization. Forward-mode automatic differentiation outperforms backward-mode automatic differentiation in computing the Jacobian matrix. It enables efficient backpropagation through the entire pipeline. The results are obtained for a single optimization iteration.

5.5. Differentiation Analysis

In our optimization pipeline, the Jacobian matrix has the property that the number of residuals significantly exceeds the number of unknowns, $M \gg N$. In this context, using forward-mode is more efficient than reverse-mode as it allows better vectorization on the GPU and eliminates the need to keep the entire computational graph in memory, which is required for revAD. This advantage is illustrated in Figure 7, where the absence of a bar indicates an out-of-memory error for revAD.

6. Limitations

While our approach enables robust real-time optimization for dynamic expressions, it does have certain limitations. Currently, the optimization pipeline operates frame-to-frame, focusing exclusively on expression tracking. This requires pre-optimizing shape parameters using existing methods. Expanding this approach to a multi-frame optimization setup could enable simultaneous tracking of both expression and shape parameters. However, this also increases the dimensionality of the resulting Jacobian matrix, which could be a challenge for maintaining real-time performance. Additionally, our method relies solely on the depth information from the Kinect sequences and does not utilize the available RGB data. Future work could use priors learned from existing in-the-wild methods [8, 9] and integrate them into the neural optimization pipeline.

7. Conclusion

In this work, we introduce OptiHead, a neural optimization pipeline that can be trained end-to-end using a novel differentiable mesh rasterizer to compute Jacobian matrices in a forward-mode automatic differentiable manner. By predicting neural residual weights and learning priors for optimization, we demonstrate robust convergence in just a few steps, enabling real-time applications. We believe this work shows how to interpolate between traditional optimization pipelines and regression techniques. However, further research is necessary to incorporate priors derived from existing methods that leverage in-the-wild data into our pipeline, ultimately enabling RGB-only tracking.

References

[1] Åke Björck. *Numerical methods for least squares problems*. SIAM, 2024. 2, 4

[2] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 157–164. 2023. 1, 2

[3] Timo Bolkart and Stefanie Wuhrer. A groupwise multilinear correspondence optimization for 3d faces. In *Proceedings of the IEEE international conference on computer vision*, pages 3604–3612, 2015. 2

[4] James Booth, Epameinondas Antonakos, Stylianos Ploumpis, George Trigeorgis, Yannis Panagakis, and Stefanos Zafeiriou. 3d face morphable models" in-the-wild". In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 48–57, 2017. 2

[5] James Booth, Anastasios Roussos, Stefanos Zafeiriou, Allan Ponniah, and David Dunaway. A 3d morphable model learnt from 10,000 faces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5543–5552, 2016. 2

[6] Aljaz Bozic, Pablo Palafox, Michael Zollhöfer, Angela Dai, Justus Thies, and Matthias Nießner. Neural non-rigid tracking. *Advances in Neural Information Processing Systems*, 33:18727–18737, 2020. 2

[7] Vasileios Choutas, Federica Bogo, Jingjing Shen, and Julien Valentin. Learning to fit morphable models. In *European Conference on Computer Vision*, pages 160–179. Springer, 2022. 2

[8] Radek Danecek, Michael J. Black, and Timo Bolkart. EMOCA: Emotion driven monocular face capture and animation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20311–20322, 2022. 1, 7

[9] Yao Feng, Haiwen Feng, Michael J. Black, and Timo Bolkart. Learning an animatable detailed 3D face model from in-the-wild images. volume 40, 2021. 1, 2, 7

[10] Simon Giebenhain, Tobias Kirschstein, Markos Georgopoulos, Martin Rünz, Lourdes Agapito, and Matthias Nießner. Learning neural parametric head models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21003–21012, 2023. 2

[11] Krishna Murthy Jatavallabhula, Soroush Saryazdi, Ganesh Iyer, and Liam Paull. gradslam: Automagically differentiable slam. *arXiv preprint arXiv:1910.10672*, 2019. 2

[12] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2, 6

[13] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020. 4

[14] Tianye Li, Timo Bolkart, Michael J Black, Hao Li, and Javier Romero. Learning a model of facial shape and expression from 4d scans. *ACM Trans. Graph.*, 36(6):194–1, 2017. 2

[15] Yang Li, Aljaz Bozic, Tianwei Zhang, Yanli Ji, Tatsuya Harada, and Matthias Nießner. Learning to optimize non-rigid tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4910–4918, 2020. 2

[16] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7708–7717, 2019. 4

[17] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 851–866. 2023. 2

[18] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019. 2

[19] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*, pages 105–116. Springer, 2006. 2

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 4, 6

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015. 3

[22] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE, 2001. 3

[23] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain. 1994. 2

[24] Jiapeng Tang, Angela Dai, Yinyu Nie, Lev Markhasin, Justus Thies, and Matthias Nießner. Dphms: Diffusion parametric head models for depth-based tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1111–1122, 2024. 5, 6

[25] Justus Thies, Michael Zollhöfer, Matthias Nießner, Levi Valgaerts, Marc Stamminger, and Christian Theobalt. Real-time expression transfer for facial reenactment. *ACM Trans. Graph.*, 34(6):183–1, 2015. 1, 2, 3, 4

[26] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2387–2395, 2016. 1, 2

[27] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999. 6

[28] Tianke Zhang, Xuangeng Chu, Yunfei Liu, Lijian Lin, Zhendong Yang, Zhengzhuo Xu, Chengkun Cao, Fei Yu, Changyin Zhou, Chun Yuan, et al. Accurate 3d face reconstruction with facial component tokens. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9033–9042, 2023. 1, 2

[29] Mingwu Zheng, Hongyu Yang, Di Huang, and Liming Chen. Imface: A nonlinear 3d morphable face model with implicit neural representations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20343–20352, 2022. 2

[30] Wojciech Zielonka, Timo Bolkart, and Justus Thies. Towards metrical reconstruction of human faces. In *European conference on computer vision*, pages 250–269. Springer, 2022. 1