Project Report

Facial Reconstruction From Video

Maxim Aslyansky (961144086), Volodymyr Polosukhin (961144094)

Abstract

The goal of the project is to develop an algorithm for face reconstruction from monocular RGB video.
Recently it has been shown that neural networks can achieve accurate results for single-image facial reconstruction, outside the scope of limited linear models.
In this project we will explore the world of 3D Face Reconstruction from video by using the following approaches:

- Extending the 2.5D output of current single-image methods with state-of-the-art geometric algorithms for dynamic 3D reconstruction to incrementally build a complete high-quality facial model.
- Experimenting with novel deep learning architectures to directly regress 3D models from images and also using recurrent neural networks to make use of temporal information.

Previous works

Unrestricted Facial Geometry Reconstruction Using Image-to-Image Translation

It has been recently shown that neural networks can recover the geometric structure of a face from a single given image. Many existing face geometry reconstruction methods restrict the solution space to some low-dimensional subspace for example by directly regressing coefficients of a 3D Morphable Faces Model. As an alternative, an Image-to-Image translation network that jointly maps the input image to a depth image and a facial correspondence map was proposed in "Unrestricted Facial Geometry Reconstruction Using Image-to-Image Translation" [viii]**.** Resulting depth and face correspondence can be then utilized to provide high quality reconstructions of diverse faces under extreme expressions, using a purely geometric refinement process consisting from geometric deformation and refinement steps. Both qualitative and quantitative analyses demonstrate the accuracy and the robustness of this approach.

DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time

DynamicFusion[i] is the first real-time dense dynamic scene reconstruction system, which succeeded to remove the static scene assumption pervasive across realtime 3D reconstruction and SLAM systems. It achieved this by generalizing the volumetric TSDF fusion technique to the non-rigid case, as well as developing an efficient approach to estimate a volumetric 6D warp field in real-time. DynamicFusion obtains reconstructions of objects whilst they deform and provides dense correspondence across time.
In this project we'll try to combine both systems in order to recover the geometric structure of a face from a video by incrementally building a detailed face model.

Convolutional mesh autoencoder

The deep learning approach is based on COMA[ii] work. In the work authors used convolutional mesh operators in spectral domain[iii] and down/up sampling using QSLIM like approach[iv] in order to encode mesh of 5023 points into 8 real numbers and decode it back with low error.

Chebyshev spectral convolution

The Laplacian of the graph is defined as $L = D - A$, where D_{ii} is degree of vertex i and A is adjacency matrix of A.

The Chebyshev polynomial of order k is given by $T_k(x) = \begin{cases} 1 & 0 = k \\ x & 1 = k \\ 2xT_{k-1}(x) - T_{k-2}(x) & otherwise \end{cases}$

The filter defined as $g_\Theta(L) = \sum_{k=0}^{K-1} \Theta_k T_k \left(\frac{2L}{\lambda_{max}} - I\right)$, where λ_{max} is largest eigenvalue of L, and Θ_k is trainable parameter and the convolution $y_j = \sum_{i=1}^{Fin} g_{\Theta_{i,j}}(L) x_i$.

Down and up sampling

Down sampling is performed by multiplication with binary sparse matrix Q_d which is constructed by contracting vertex pairs iteratively by minimizing surface error approximation. And up sampling matrix Q_u is built at the same time. During down sampling original vertex mapped into down sampled surface by projecting it into closest triangle and computing barycentric coordinates, the up sampling weights are chosen respectively.

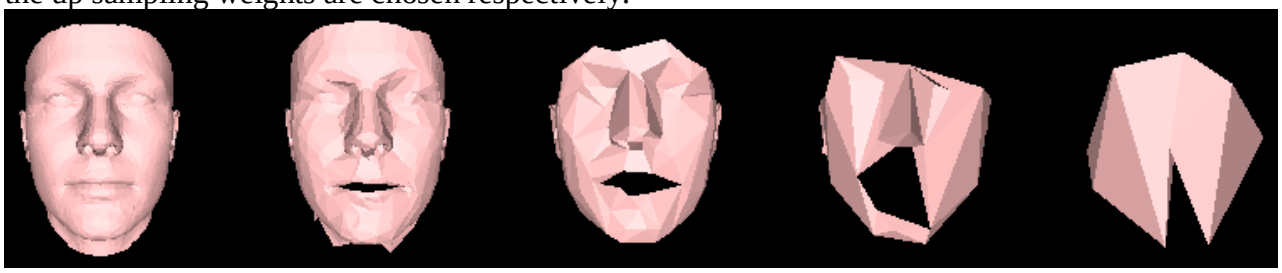


Figure 1. QSlim mesh down sampling

Preprocessing

In articles authors worked on low resolution meshes which they standardized vertex wise (the latter fact appeared to be crucial for learning).

Morphable Face Models - An Open Framework

Basel Face Model BFM-2017

One of the contributions of the work[v] is PCA basis of face shape (S), expression (E) and colors (C). Now if we denote as $\cdot_b, \cdot_\mu, \cdot_\sigma$ the basis, mean and standard deviation of PCA basis, then points of new mesh could be generated as $\left(S_\mu + c_s \cdot S_\sigma \cdot S_b\right) + \left(E_\mu + c_E \cdot E_\sigma \cdot E_b\right)$ and colors as $C_\mu + c_c \cdot C_\sigma \cdot C_b$. And if with c_s, c_C, c_E changing smoothly in time, one can generate a mesh video.

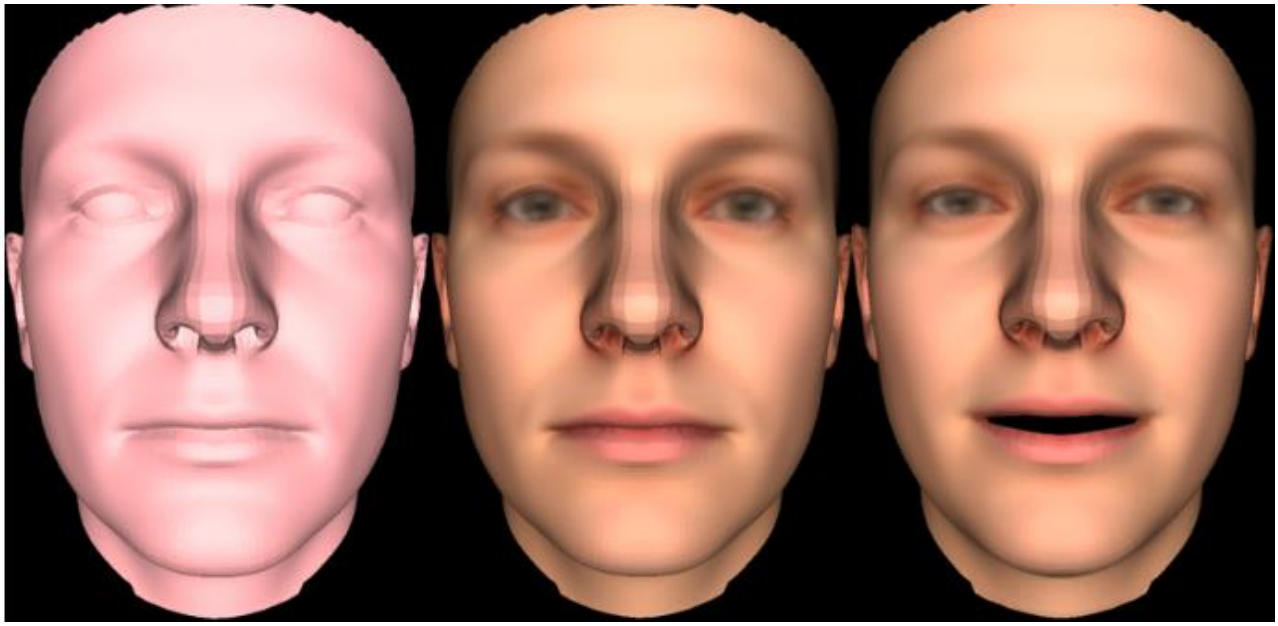


Figure 2. From left to right: shape, shape with color, shape with color and emotions

Unsupervised Training for 3D Morphable Model Regression

Differentiable, 3D mesh renderer using TensorFlow

One of contributions of an article[vi] is implementation of renderer[vii] which allowed us to create input synthetic images on the fly for training and validation and, what is more important, allowed us to add “rendering loss”.

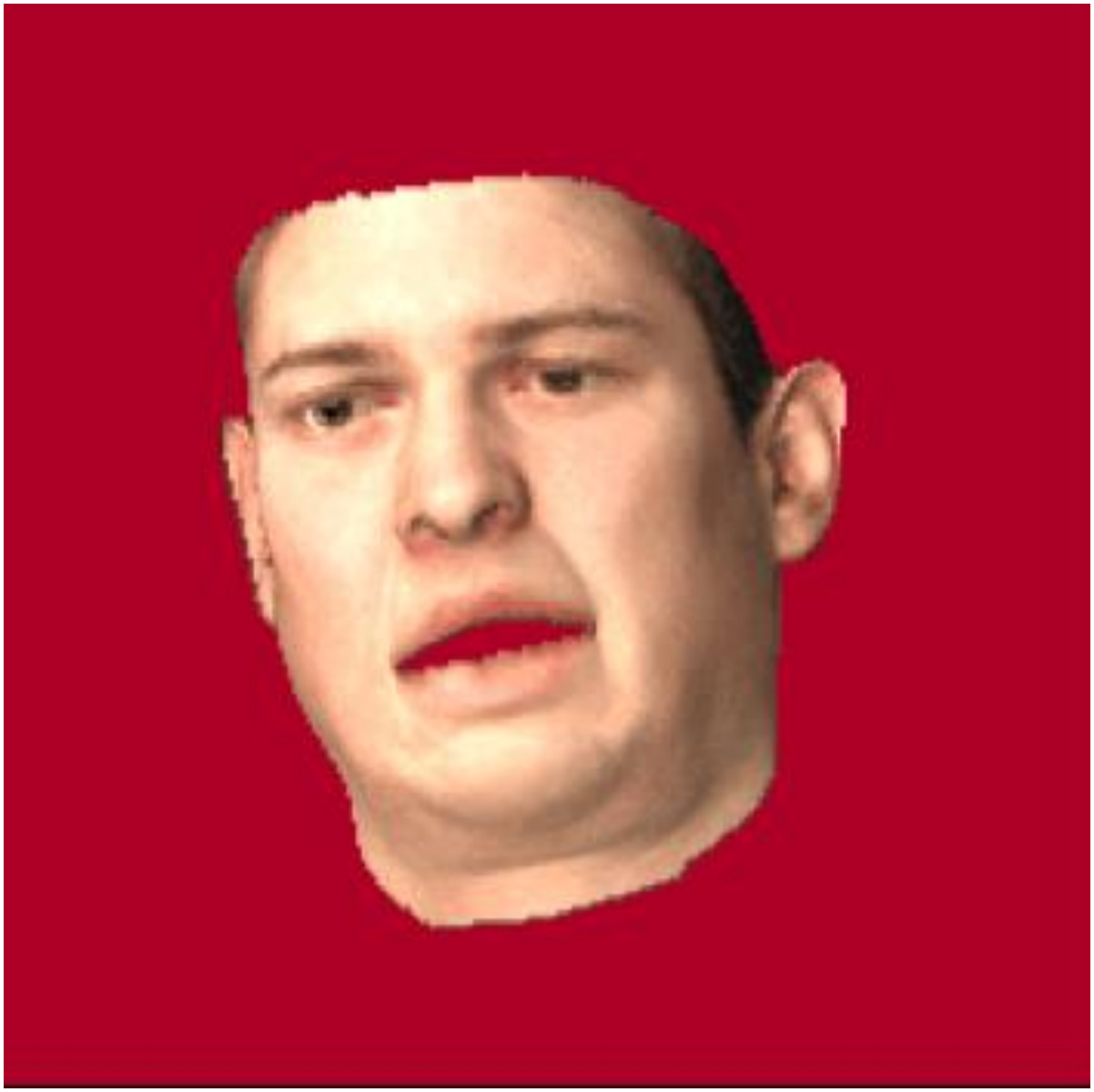


Figure 3. Rendered facial mesh

Solution

“Unrestricted Facial Geometry Reconstruction” applied to video

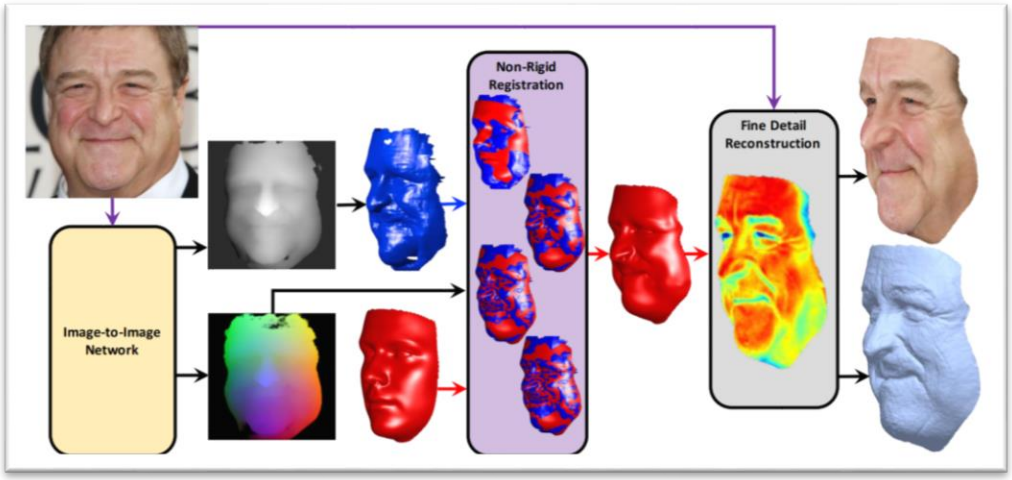


Figure 4. Algorithmic pipeline

The algorithmic pipeline of the system is presented in Figure 4. Algorithmic pipeline The input of the network is a facial image, and the network produces two outputs: The first is an estimated depth map aligned with the input image. The second output is a dense map from each pixel to a corresponding vertex on a reference facial mesh. To bring the results into full vertex correspondence and complete occluded parts of the face, we warp a template mesh in the three-dimensional space by an iterative non-rigid deformation procedure. Finally, a fine detail reconstruction algorithm guided by the input image recovers the subtle geometric structure of the

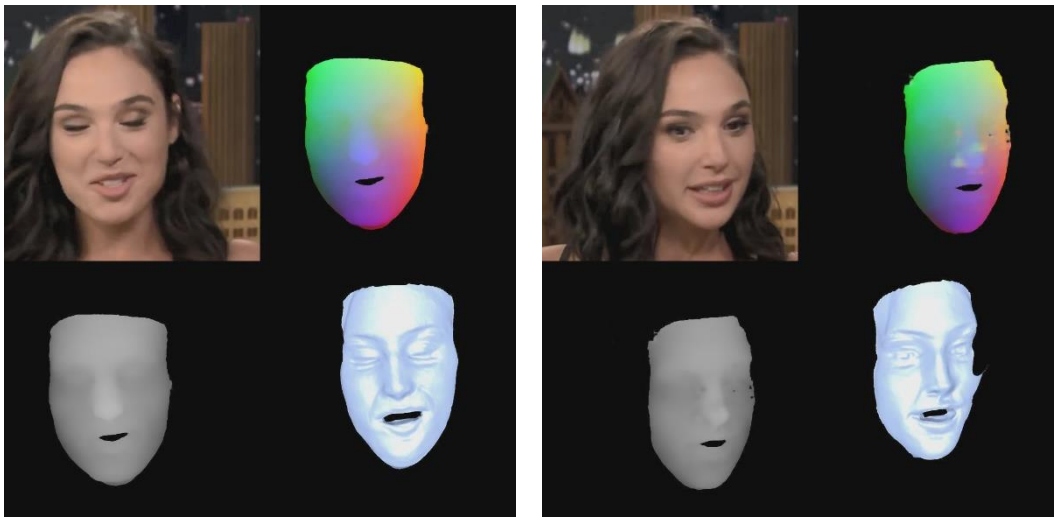


Figure 5. Recursive NRR

face.

We started our work by using the most straightforward approach to extend the previous pipeline to the video. More specifically, we used original network to obtain dense per-pixel depth and face correspondence.
First depth frame is processed similarly to the original approach where we try to deform a known template to a mesh obtained by connecting neighboring pixels of depth frame. When next frame arrives, instead of using the same original template we use previous output mesh for this task. Since subsequent frames contain the same face, usually with only slight non-rigid motion, this approach helps to dramatically reduce number of iterations needed for Elastic Deformation algorithm to converge. Furthermore, we found that this way the final meshes behave much smoothly in temporal domain producing more perceptually pleasing results. The downside of the approach is accumulative mesh deformation artifacts which lead to divergence due to numerical inaccuracies. In this case template mesh suddenly "explodes" and we are forced to reset template mesh to the original one and start NRR algorithm with more iterations. Our results presented in Figure 2 – successful frame reconstruction on the left as well as frame with mentioned deformation artifacts on the left.
Fine detail reconstruction algorithm is left without changes and is applied on each mesh one-by-one.

Further improvements to recursive NRR

After identifying and analyzing the problems of previous approach we decided to design and implement some algorithmic improvements to increase its accuracy and robustness.
Firstly, we refactored the code and optimized it for better performance (e.g. removing duplicate code, optimizing, playing with parameters) so now all the steps except of NRR take less than a second.
We found that many of the issues of previous approach were caused by partially inaccurate or noisy output of Image-to-Image network.

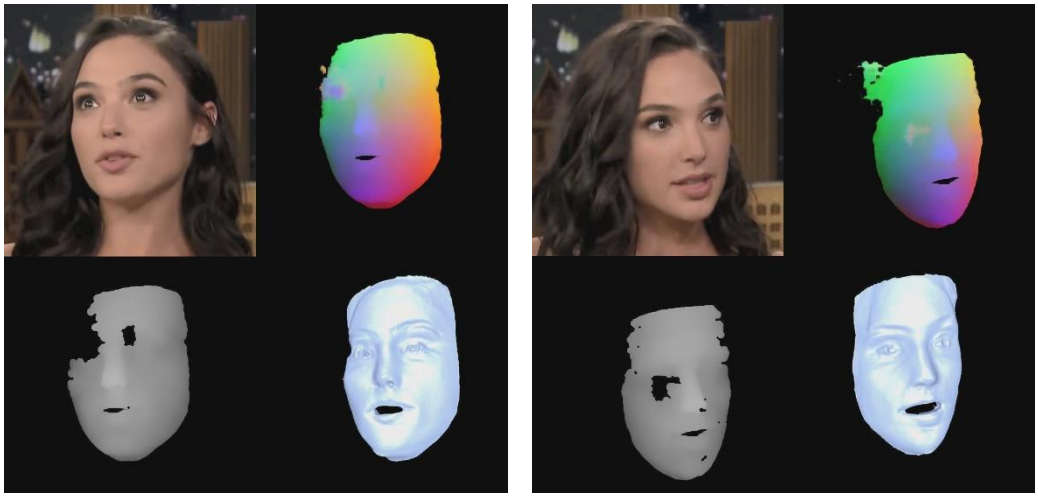


Figure 6. Our depth filtering and corresponding reconstructions

In order to tackle the issue of invalid input to NRR (such as strange blobs outside the face or distorted values inside it) we developed a simple algorithm to detect and discard the invalid network output.
The algorithm builds mask based on smoothness of x and y correspondence map estimated by the

network – ideally they should be spatially smooth (nearly planar), so we can detect inaccuracies by applying thresholding to these coordinates filtered by a Laplacian-of-Gaussian kernel. We found this approach to perform reasonably fast and consistently. Since in our case it's more important to preserve correctness of canonical model (rather than trying to preserve as much detail as possible) we used a relatively low threshold to discard all the outliers with high confidence.
This results in target mesh which is often half-empty but contains only valid vertices. Registering partial meshes requires special handling so we changed NRR the algorithm accordingly.
First we detect border vertices on the target mesh and store them. All template vertices that correspond to them will be "frozen" which means we will increase their membrane term, so that they'll move rigidly relatively to their neighbors.
Additionally, we introduced more robust matching scheme: when looking for pncc/coordinate correspondences we discard ones with high forward-backward errors. We begin by nearest neighbor search in target mesh for each template vertex. Then we search for nearest neighbor of previously found vertices and look for distance between the result and initial mesh: if it's higher than some threshold we "freeze" these vertices as well.
We also improved affine alignment by introducing the forward-backward trick, which helps to remove outliers. Then we added an option to iteratively fit the transform using all inliers (instead of minimal set in RANSAC) which gives better results on average and converges more quickly.
The results of our new approach are presented in Figure 3. You can see 2 frames were incorrect depth values are identified and then modified Elastic Deformation algorithm successfully reconstructs the mesh.
Although our modifications significantly improved the robustness and also speed of the original approach, we found them not sufficient neither for real-world scenarios nor for real-time operation. The main performance bottleneck is decomposing and solving formulated energy problem which contains approximately 3*25k variables. Current implementation needs 5 seconds to only calculate LU decomposition which must be done several times for each frame.
This leads us to conclusion that decreasing the number of estimated variables is crucial for reasonable performance. We'll address this issue in the next section.

Moving to Dynamic Reconstruction approach

As we already mentioned DynamicFusion was the first system to successfully perform non rigid dynamic reconstruction in real-time. We started by looking for some open source implementations of the algorithm. After integrating the code to our pipeline, we got the results depicted in Figure 4:

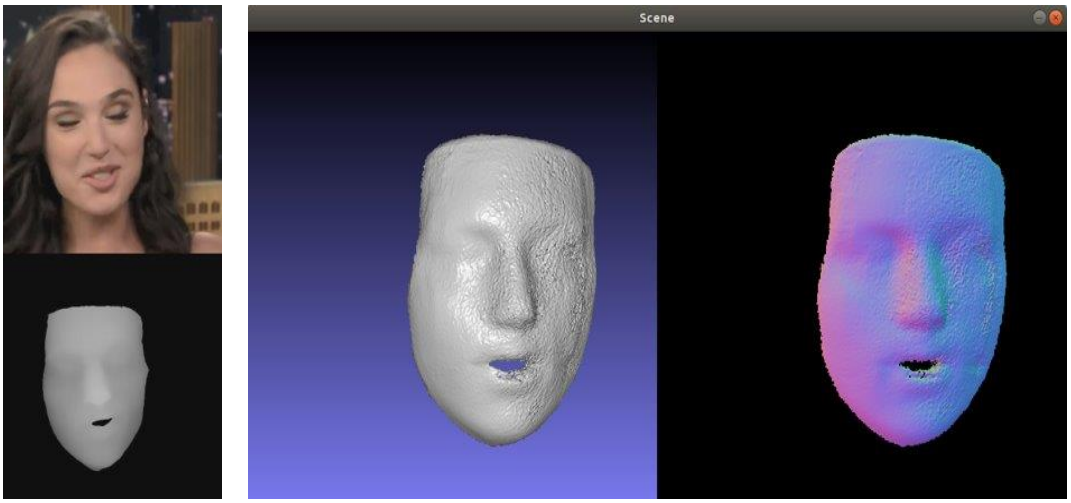


Figure 7 - open source DynamicFusion implementation

As can be seen, there are some visible artifacts in the reconstruction. In our opinion it can be caused by failing to correctly estimate the warp field which leads to misalignment artifacts smoothed by TSDF fusion. The code was also almost 1000x slower than stated in the paper and incredibly difficult to get working.

Modifying and implementing DynamicFusion for Facial Reconstruction

After previous attempt we decided to implement the algorithms by ourselves.
We also identified some important changes to the original algorithm which would help to simplify the implementation and increase frame-rate:

- Since we have some prior knowledge about the scene, we can use a known template as a starting canonical model
- This enables us to calculate Embedded Deformation and Regularization graphs only once and re-use them for each new frame
- Additionally, in this case there's no need to use TSDF fusion to accumulate new data as the meshes can be added by simply fusing transform graphs
- This also removes the need to implement TSDF calculation and Marching Cubes algorithm in order to move from voxel representation to mesh and vice versa.

The first component of the algorithm is Embedded Deformation approach in which instead of estimating per-vertex transformation parameters we define a sparse graph of transformation nodes. Then each vertex is transformed using weighted average of nearest nodes. Similarly to the original paper, we defined the graph of dual quaternions and used Dual Quaternion Blending for per-vertex calculation.

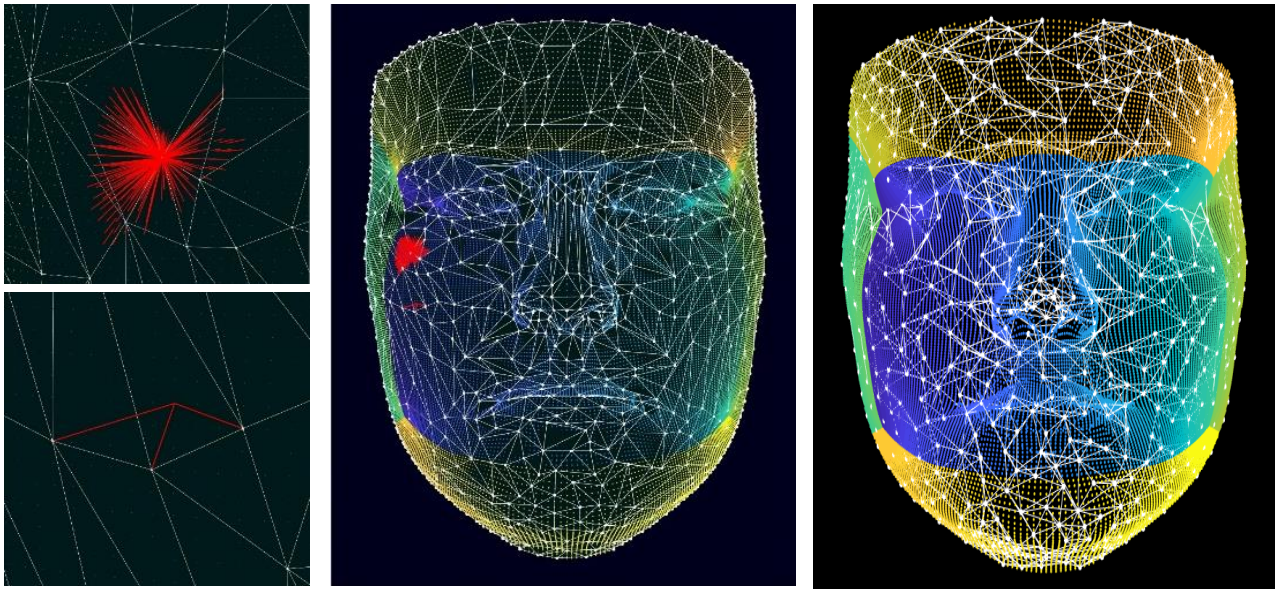


Figure 8. Embedded deformation graph built by mesh decimation (center) and farthest point sampling (right)

Figure 5 illustrates two approaches to building Embedded Deformation graphs: mesh decimation which aims to minimize geometric distortion of resulting mesh and farthest point sampling which tries to maximize the minimal distance between each pair of nodes. We found the later approach to perform much better since it leads to more uniform coverage of the mesh and unlike mesh decimation doesn't have a lot of redundant nodes at the edges. Then we used knn search to define regularization graph. In our experiments we found that 700 is optimal number of nodes to achieve best performance / accuracy tradeoff.

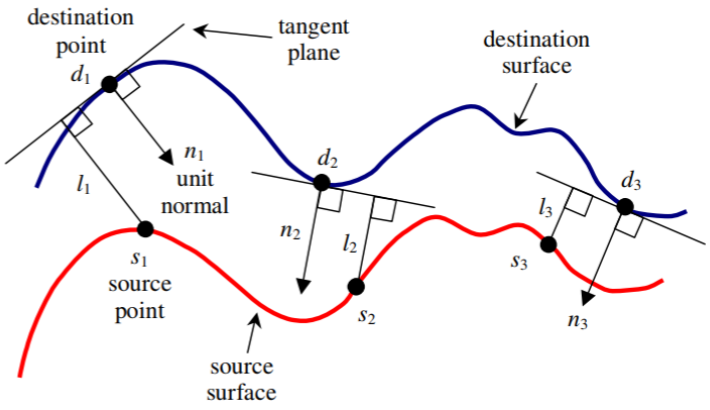


Figure 9. Data terms illustrations

After this step, we proceed to defining different energy terms similarly to DynamicFusion.
Data terms are:

- Point term: $r_{pt-pt,n} = (d_i - s_i)_n$
  Although this term is not present in the original algorithm, we found that it improves convergence
- Point-to-Plane term: $r_{pt-pl} = (d_i - s_i) \cdot n_i$
- Also, in future color/pncc correspondence terms could be added

In any case, noise, missing data and insufficient geometric texture in the live frame – an analogue to the aperture problem in optical-flow – will result in optimisation of the transform parameters being ill-posed. How should we constrain the motion of non-observed geometry? Whilst the fully correct motion depends on object dynamics and, where applicable, the subject's volition, we make use of a simpler model of unobserved vertices by enforcing spatial continuity with corresponding regularization terms:

- Smooth Transformation term
  Similarly to DynamicFusion we regularize difference between neighboring nodes' transforms using L2 norm
- Conformal Mapping term
  In original DynamicFusion implementation Dual Quaternion Blending was used were the final transforms are normalized by quaternion's norm. We decided to remove this normalization similarly to recent Fusion4D paper as it improves convergence and reduces computations. In this case a scale DoF is added to the (original) rigid transform. Instead we add a soft regularization term which forces quaternion's norm to be close to 1.
- Levenberg-Marquardt dumping term

The main computational complexity in minimizing E involves constructing and factorizing the Gauss-Newton approximation of the Hessian:

$$(J^T \cdot J + \lambda \cdot I) \cdot \delta = J^T \cdot f$$

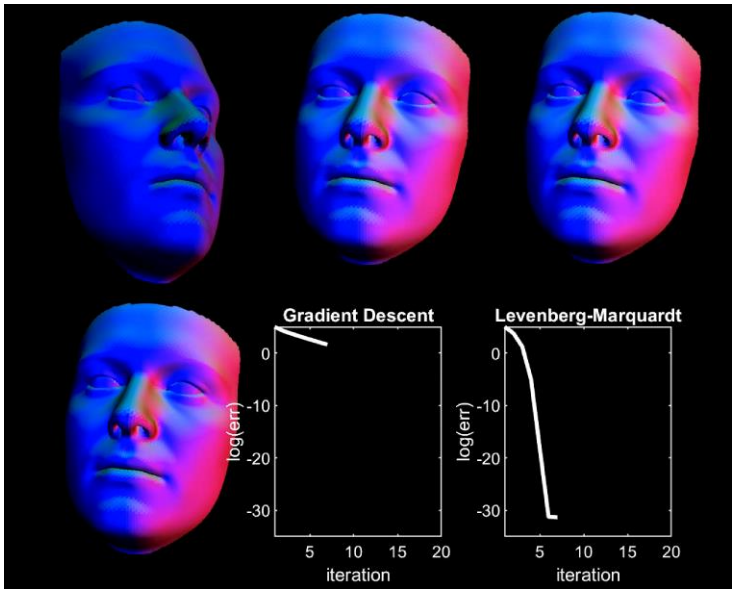


Figure 10. Levenberg Marquardt algorithms converges much quickly compared to gradient descent

We used several techniques to improve the speed:

- Evaluating $J^T \cdot J$ and $J^T \cdot f$ in place instead of explicitly storing them in the memory
- Similarly to DF algorithm we neglect off-block-diagonal data terms
- Most operations are highly parallel, so our vectorized Matlab implementation achieves reasonable frame-rate (almost 25x faster than previous approach) mainly due to lees overhead of matrix solver
- We also implemented significant part of the algorithm to be run on GPU using CUDA framework and interfaced it to Matlab with CudaMex. This indeed gave us significant performance boost and we saw potential for real-time performance. Unfortunately we got stuck on trying to implement NN-search on GPU and later found it increasingly difficult to mae algorithmic changes to GPU code so we decided to completely move to Matlab branch of the algorithm.
  We believe, that in future our work could be finished to achieve real-time frame-rates.

Results of the final algorithm are in Figure 8:

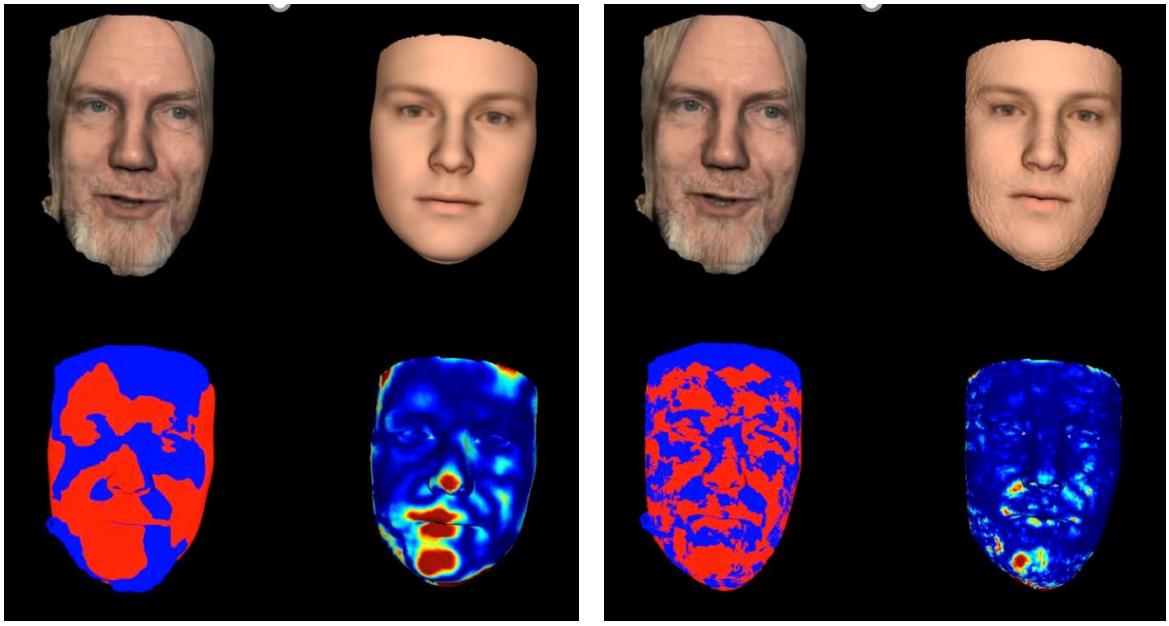


Figure 11. Our DynamicFusion implementation after a couple iterations and near the convergence

We also found it helpful to dynamically decrease smoothness term similarly to Elastic Deformation implementation provided by Matan and Elad, as this helps to avoid getting stuck near local minima after first iterations.
As can be seen our algorithm achieves convergence although some artefacts are visible as well.

Figure 12. Our algorithm applied to a video

After implementing a single-frame alignment scheme we decided to extend the algorithm to video by initializing node transform by values from the previous iteration.
We also found it helpful to initialize the template by result of the original recursive NRR applied to the first frame.
Figure 9 shows a successful frame on the left as well as a failure case on the right.

After evaluating the new algorithm on our test set we came to conclusion that the new approach doesn't provide significant improvement in terms of perceptual quality. From the other hand it is much faster which makes it significantly more suitable for practical applications.
Additionally, the usage of non-linear optimization could help us to introduce more data terms and also add robust regularizers (instead of L2 norm).
We'll address this points in the following section.

Picture to vertex

We used encoder decoder model like suggested in COMA, replacing mesh encoder with picture encoder, increasing latent space from 8 to 165 and few other modifications, until network started to work.
Since QSlim implementation worked unbearably slow on full BMM mesh size, we down sampled mean shape with MATLAB reducepatch function. On the beginning we worked on mesh with 4096 triangles and after debugging moved to 25000 triangles.
As the input we used images of shape 224x224x3, generated from mesh before down sampling. We started by stacking 3 pictures channel-wise and eventually after increasing of network size succeeded to regress mesh from single picture. The rendering parameters (rotation, translation, illumination, background, focal distance) were generated randomly. Here we should mention the clear miss of using single colored background.

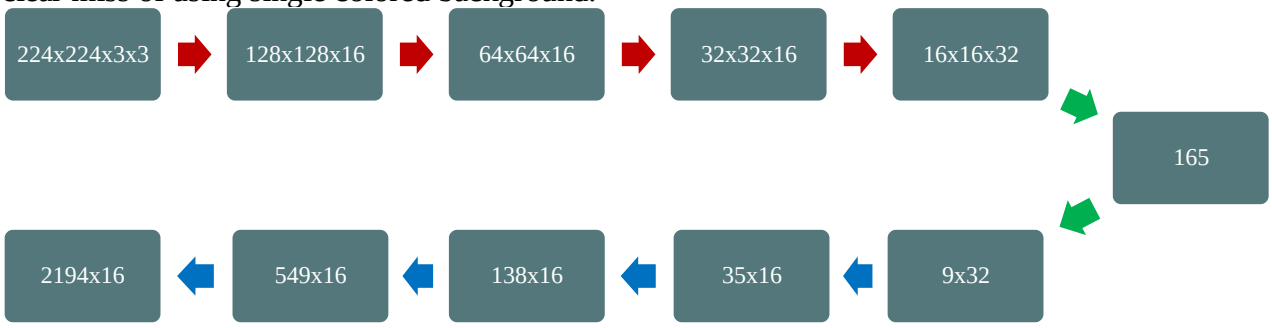


Figure 13. Network used for mesh with 4096 faces. Red arrows are image convolutions followed by down sampling. Green arrows are fully connected layers. Blue arrows are mesh up samplings followed by mesh convolution.

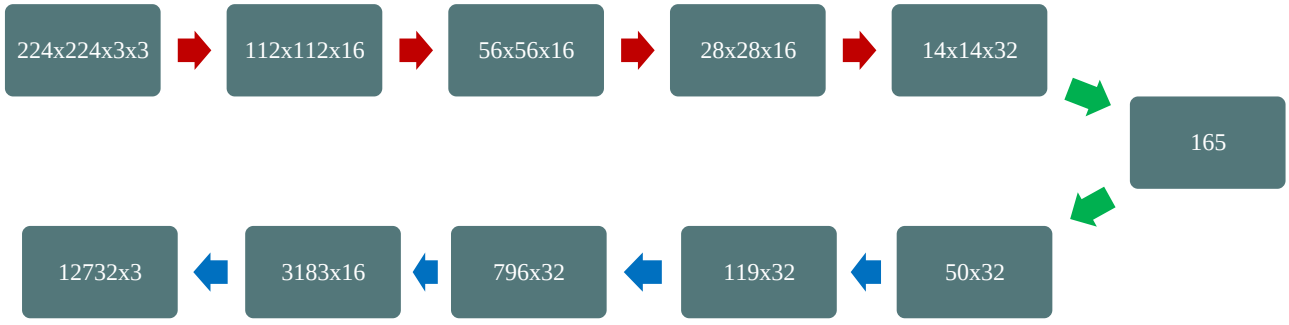


Figure 14. Network used for 25000 faces.

Clear advantage of encoder-decoder architecture is its extendibility to predict new targets. For example, on some stage we added rotation prediction just as another decoder "tail" from same latent space.

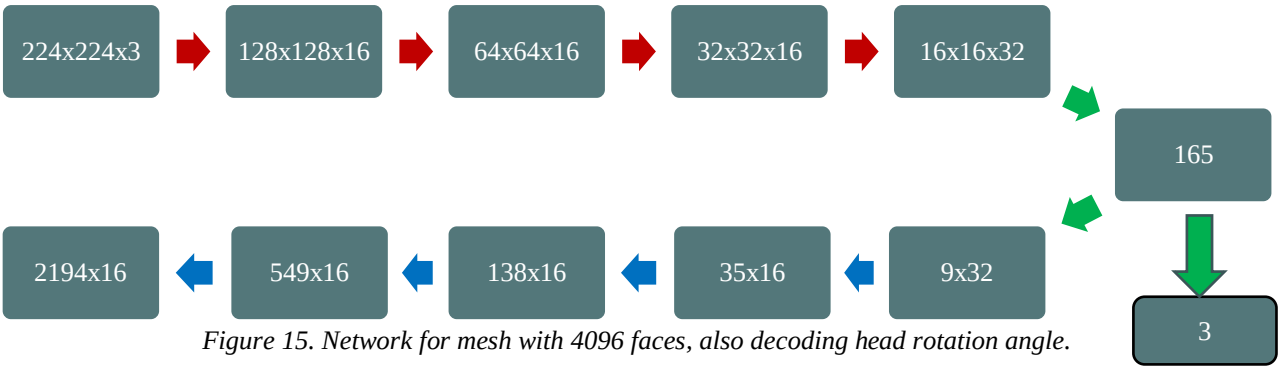


Figure 15. Network for mesh with 4096 faces, also decoding head rotation angle.

Video to vertex

In order to generate realistic video, we smoothly changed coefficients for PCA basis and also smoothly changed rendering parameters (rotation, translation, illumination, background, focal distance).
We upgraded the previous model by adding LSTM layer between encoder and decoder. Basic assumption that with frames progress recurrent neural network is going to remember long time information about the identity from previous frames and forget irrelevant information about the expressions.
For some reason, when we tried to train RNN end-to-end the training did not converge, so we had to initialize encoder and decoder with weights from picture to vertex model.

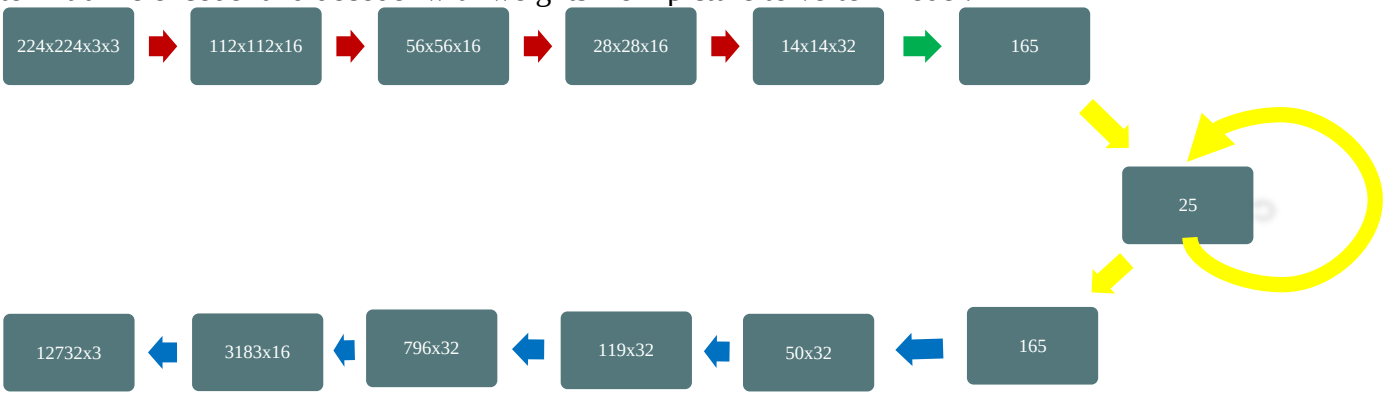


Figure 16. Recurrent network for mesh reconstruction

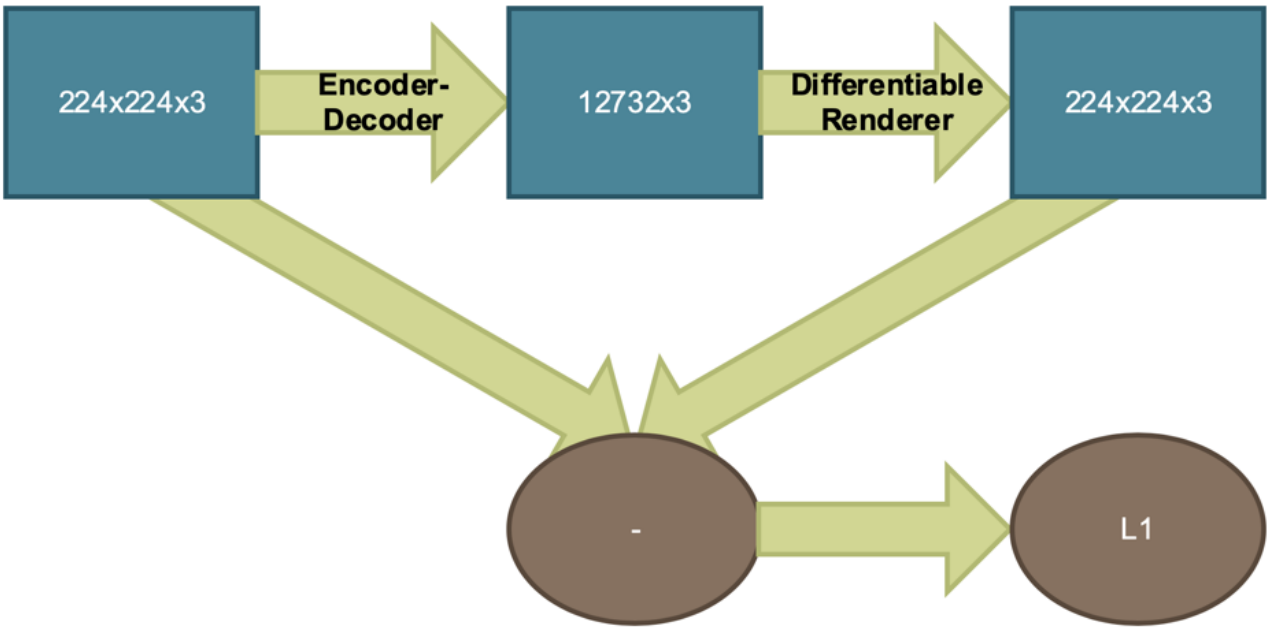
Normal loss

Since the resulting mesh often had some high frequency noise, inspired by an article[viii] we added normal loss to network. We computed it by destandardizing output and ground truth mesh, computing for each vertex in each triangle ΔABC unit normal vector $\frac{(B-A)\times(C-A)}{\|(B-A)\times(C-A)\|}$ and taking mean absolute difference of the corresponding coordinates.

Figure 17. Normal to mesh

Rendering loss

The reasonable thing to demand from picture to vertex network is that the output when rendered back with the same parameters as it was generated should be close to input. This was not always the case before, so we decided to add this loss.



Results

Picture to vertex

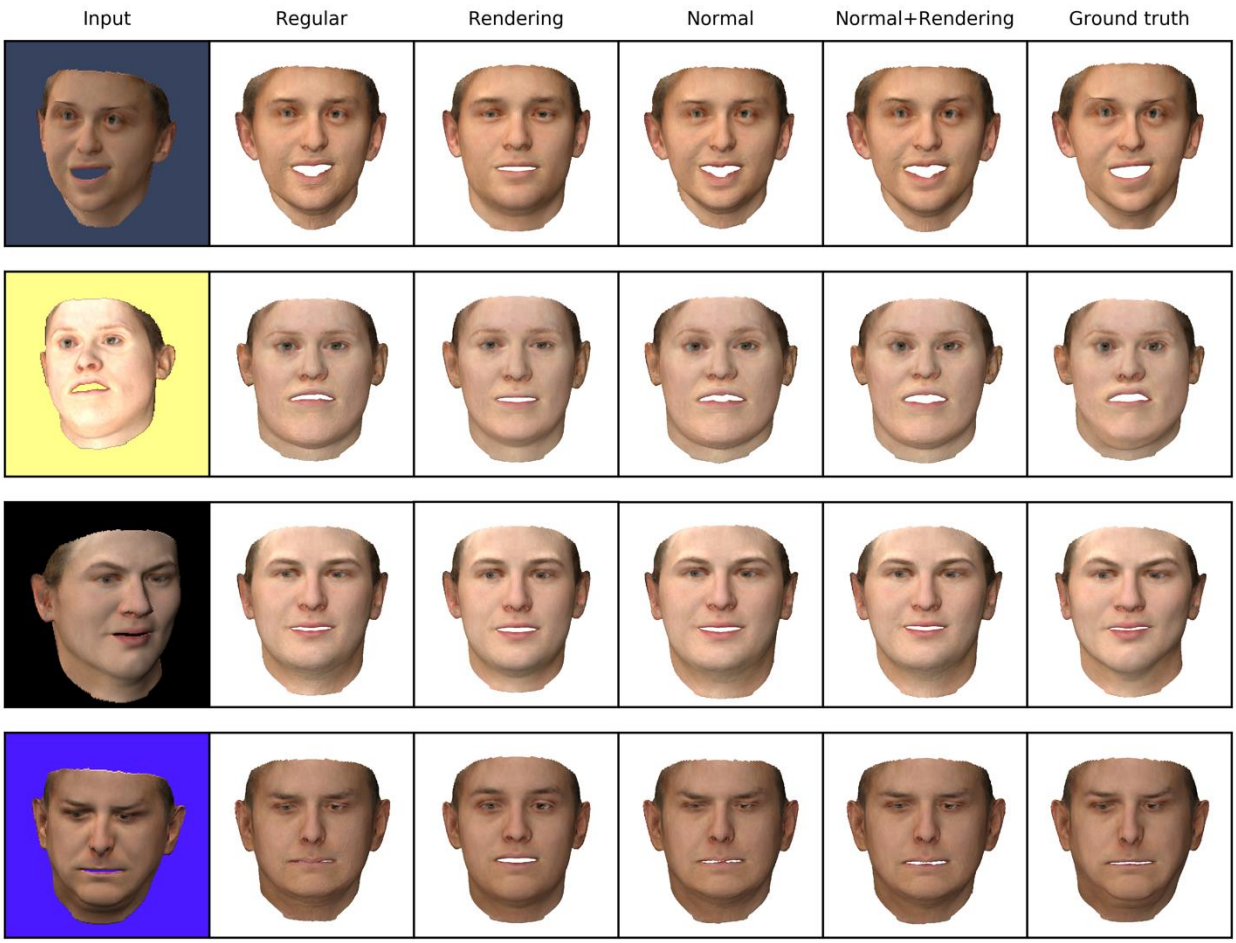


Figure 18. Visual comparison between models trained with different losses

Table 1. Error (in units of BMF) comparison between losses

	MAE	Median absolute error	RMSE	R^2	Std absolute error	90% absolute error
Regular	1.91	1.23	2.50	0.9989	1.62	4.05
Rendering	3.24	1.59	4.27	0.9969	1.62	4.05
Normal	1.89	1.22	2.47	0.9989	1.62	4.05

	MAE	Median absolute error	RMSE	R^2	Std absolute error	90% absolute error
Normal+Rendering	1.91	1.23	2.50	0.9989	1.62	4.05

Video to vertex

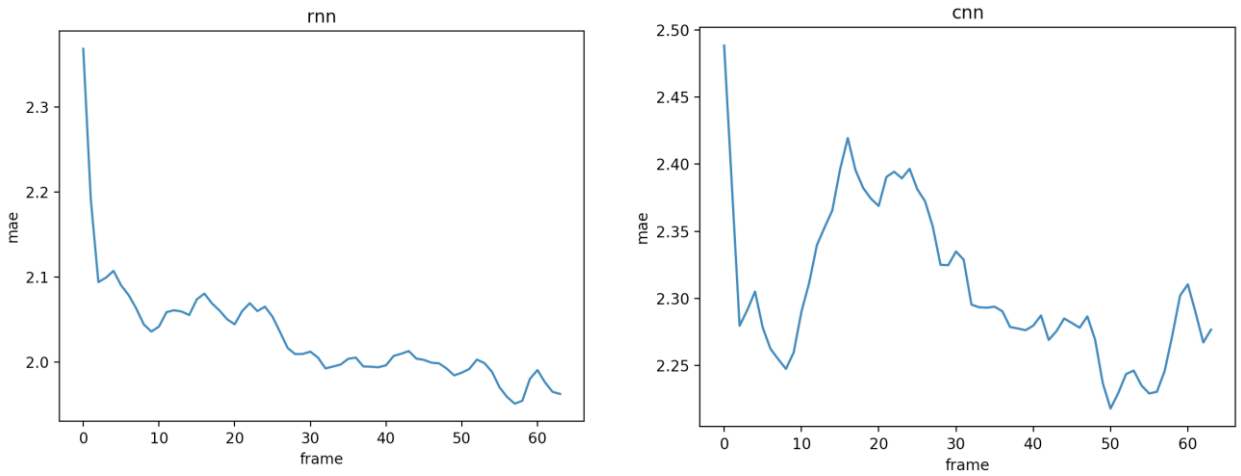


Figure 19. Mean absolute error as a function of frame in RNN (left) and CNN (right)

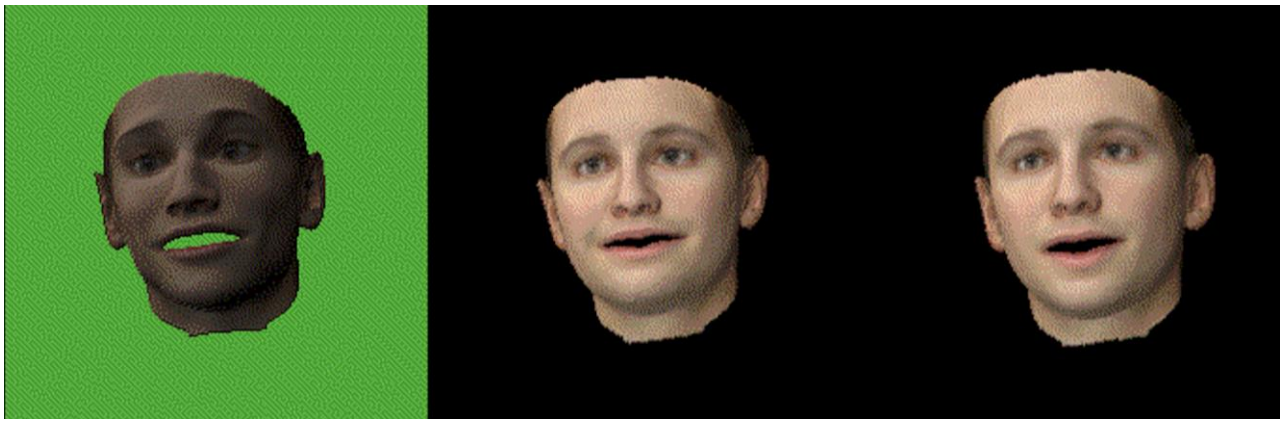


Figure 20. Input, CNN and RNN outputs

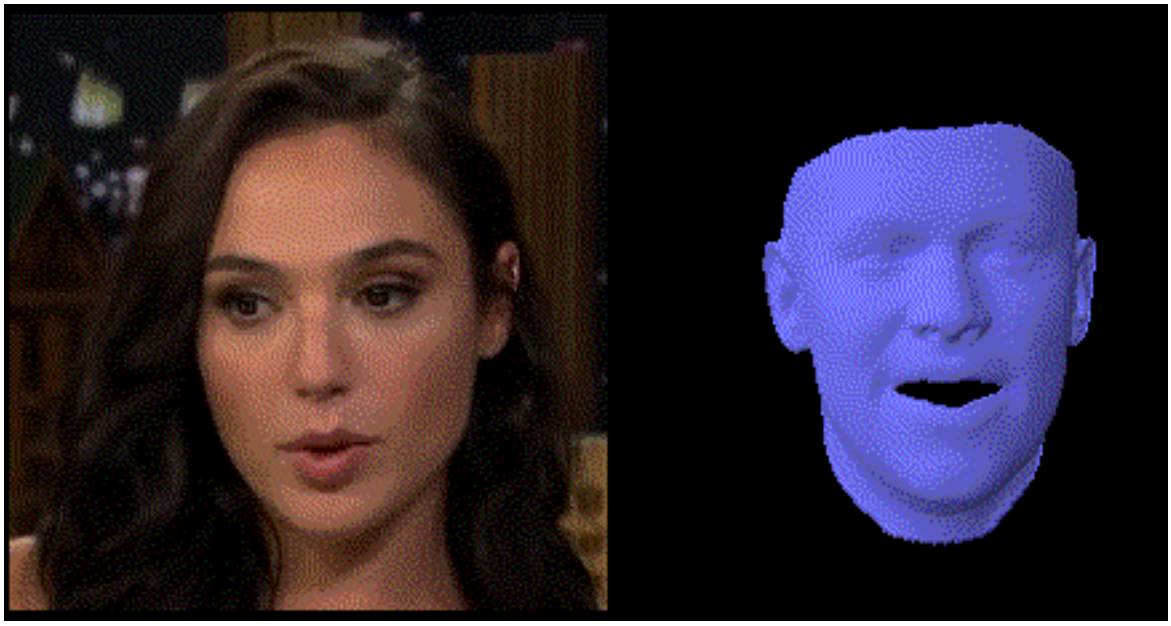


Figure 21. Model when applied to real world data

Future Work

Dynamic reconstruction

Although we achieved significant improvements compared to original "vanilla" algorithm applied frame-by-frame and our straightforward implementation we believe some modifications could be done to further improve our solution:

- First our improvements to recursive NRR such as robust matching and boundary vertices "freezing" should be integrated to DF approach
- Robust cost function such as Huber loss should be used instead of L2 norm, especially for regularization
- Explicit color or pncc losses should be implemented (exploiting the ability to add non-linear losses)
- Visual tracking or facial landmark detection should be used to ensure better temporal smoothness
- CUDA version of our code should be finished to achieve real-time performance

Neural network

The neural network trained on synthetic data indeed captures some expressions on synthetic data dataset, but totally fails when applying on real worlds data. The possible directions to improve the architecture are:

- Separate identity from expression
- Predict colors
- Use real data
    - BU-4DFE
    - BP4D
    - 3DMAD
    - Florence

- CoMA

Conclusions

Dynamic reconstruction

We presented an unrestricted approach for recovering the geometric structure of a face from a video. We demonstrated significant improvements in terms of speed, accuracy and perceptual quality compared to single-image methods. Then we further improved our approach by implementing a custom dynamic reconstruction algorithm. We believe that after some additional modifications our work could be useful for a wide variety of real-world applications.

Neural network

The picture to vertex model is indeed capable of capturing some features, mostly expressions, but it smoothens them and the result is noisy and far from perfect.
The normal loss indeed helps to reduce the noise. The rendering loss together with normal improves the visual result a bit, but without normal loss model does not learn at all.
The RNN indeed learns some internal state, since its error is decreasing with number of frames, and even smooths some artifacts. The learning of RNN end-to-end on current data is problematic, we had to initialize it with pretrained CNN.

---

[i] R. A. Newcombe, D. Fox and S. M. Seitz. DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time

[ii] A. Ranjan, T. Bolkart, S. Sanyal and M. J. Black. Generating 3D faces using Convolutional Mesh Autoencoders

[iii] M. Defferrard, X. Bresson and P. Vandergheyns. Convolutional neural networks on graphs with fast localized spectral filtering

[iv] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics.

[v] T. Gerig, A. Morel-Forster, C. Blumer, B. Egger, M. Luthi, S. Schonborn and T. Vetter. Morphable Face Models - An Open Framework

[vi] K. Genova, F. Cole, A. Maschinot, A. Sarna, D. Vlasic and W. T. Freeman. Unsupervised Training for 3D Morphable Model Regression

[vii] https://github.com/google/tf_mesh_renderer

[viii] M. Sela, E. Richardson, R. Kimmel: Unrestricted Facial Geometry Reconstruction Using Image-to-Image Translation