u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva

Computer Vision Lab_6 Report:

1.2 First Part:

We implemented the rigid factorization code in the file that we submitted. You can find our implement in the file “rigidfactorization_ortho.m” file. The function takes as the input 2D tracking data from a monocular video with dimension 2F X P. Dimension F represents number of the frames whereas P represents the number of points. We have two outputs for this function. First one is M [2FX3] which is camera rotation matrix per frame f. Second output is S [3 X P] which represents the rigid shape.

For the task two we run the file “main_rigid.m” to detect output images and detect 3D reconstruction error. Here are the results that we have obtained:

Resulting the estimated 3D reconstruction error: 0.002865 %

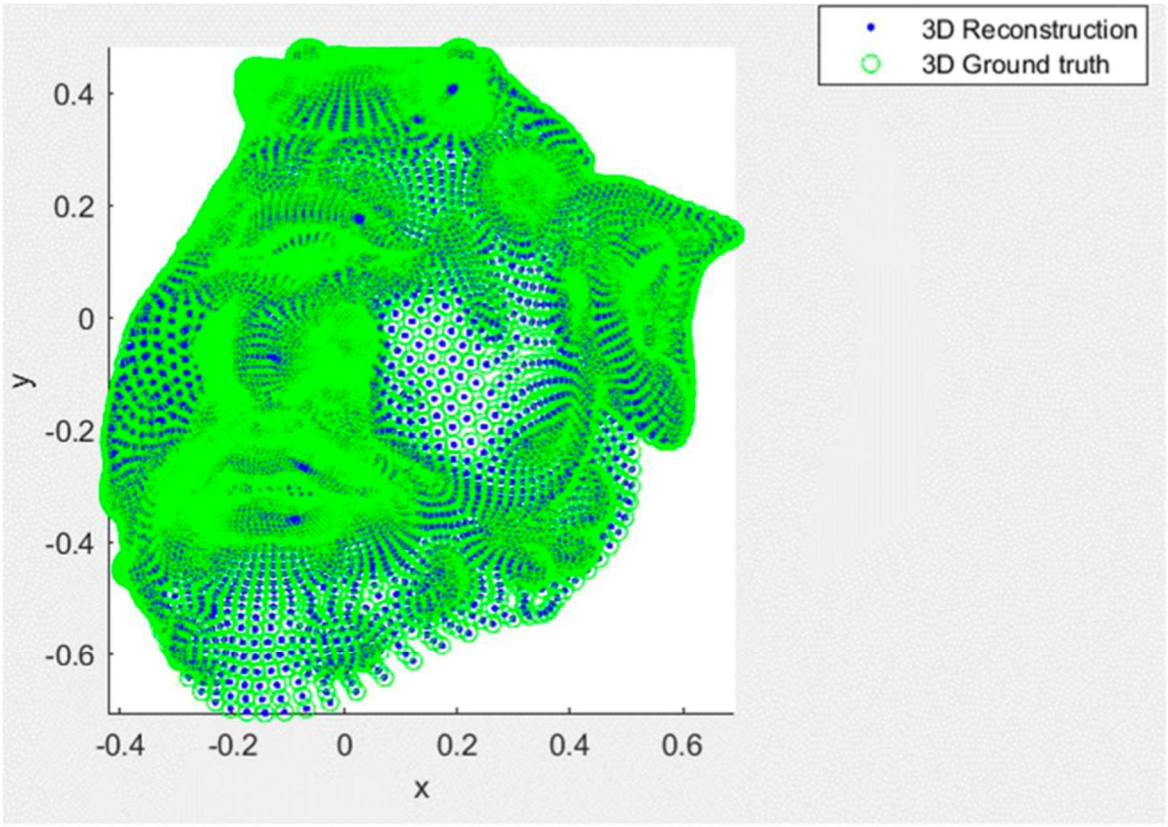
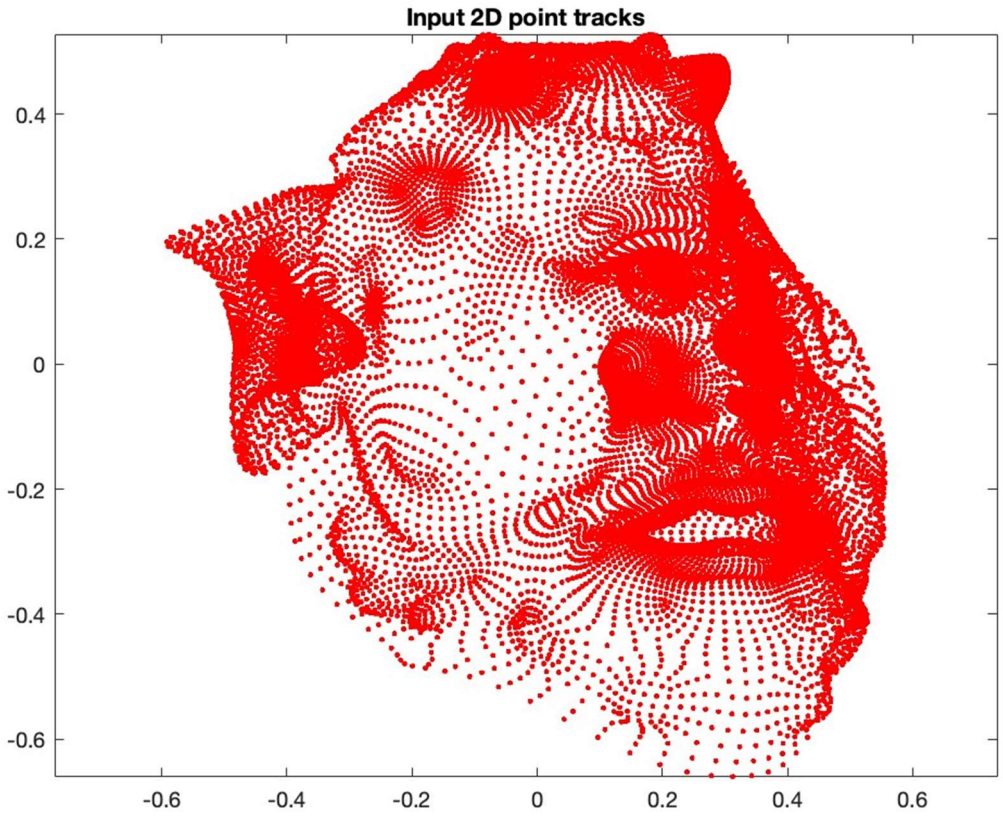


Figure 1: Comparative output of the 3D reconstruction algorithm on provided example

u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva



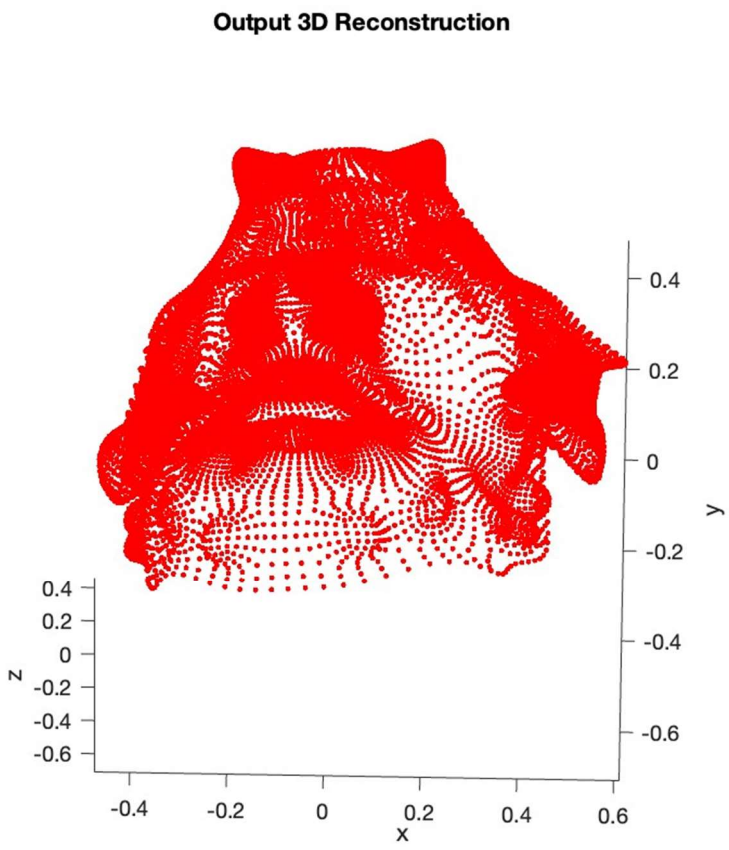


Figure 3: Output of the 3D reconstruction algorithm on provided example

1.3 Second Part:

Goal: Practice on structure from motion by assuming the shape is non rigid. The considered camera is "orthographic camera", then the solution is obtained by non-linear optimization. Here are the provided files:

- nrsfm.m: The main script to infer time-varying shape and motion from a measure- ment matrix.
- JacobianPattern.m: This file implements a function to encode the binary Jaco- bian pattern.
- cost RXT.m: This file implements the cost function as a combination of the data term and the priors whether they are used.
- model2vector.m: This file implements a function that takes as input the matrices for all parameters (camera rotations, camera translations, shape basis, and weight coefficients) and transform this information into a vector. This function represents the inverse step of vector2model.m.
- vector2model.m: This file implements a function that takes as input a vector with all variables we have to estimate, and transform it into a set of model matrices. This function represents the inverse step of model2vector.m.

u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva

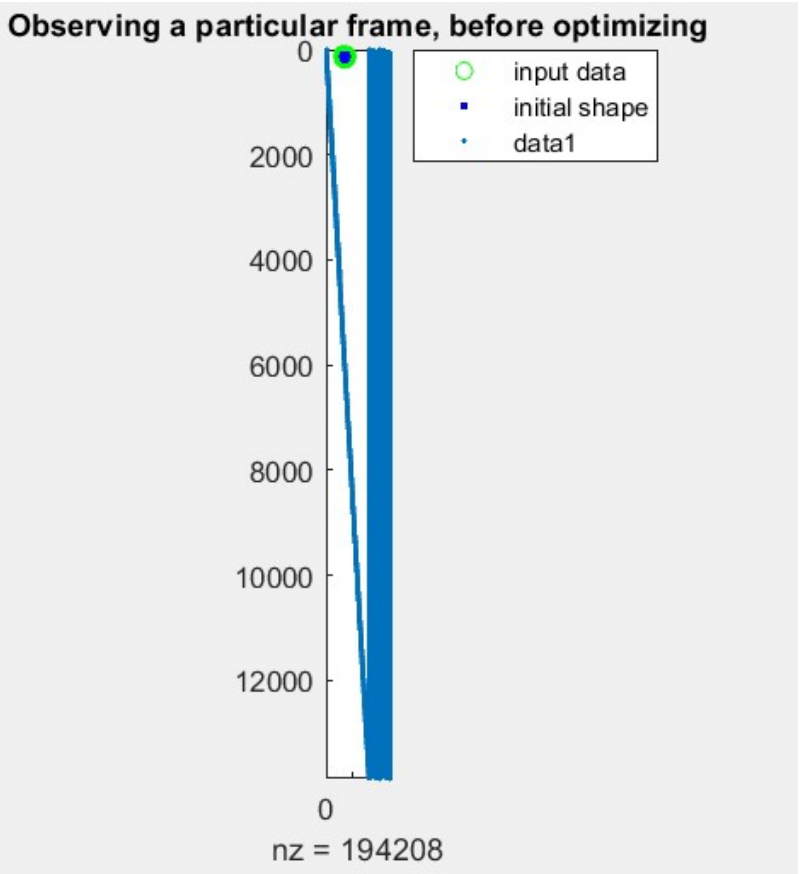
JacobianPattern.m:

In this part, we implemented the JacobianPattern function. You can find our implementation in the provided "JacobianPattern.m" file. Here is the input and output details of the function:

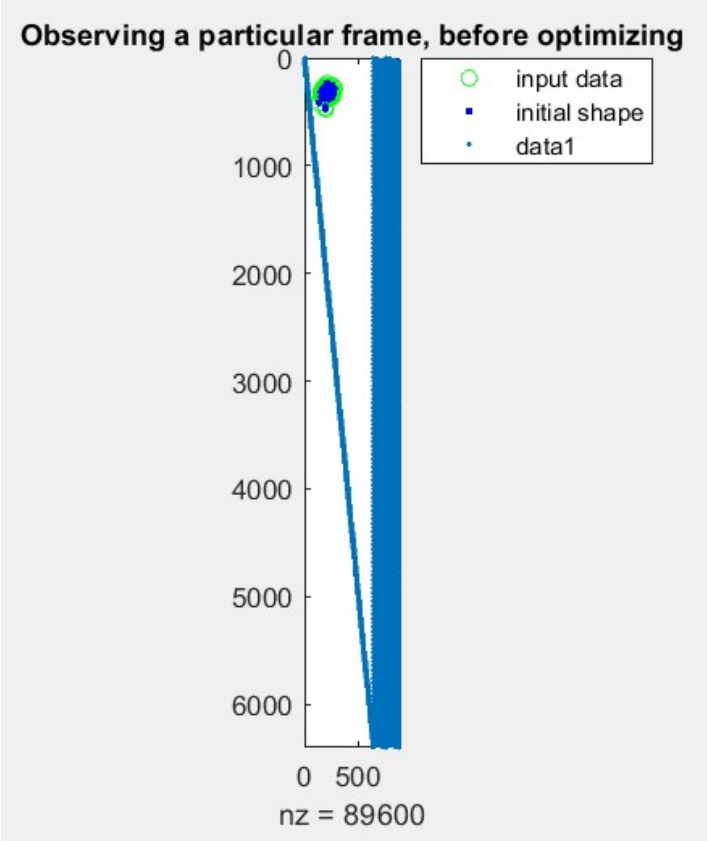
```
function [J]=JacobianPattern(K,n_frames,n_points,vij,priors)
Input
K: shape basis rank
n_frames: number of frames
n_points: number of points
vij: visibility map
priors: structure with fields:
      priors.camera_prior: boolean, 1 for rotation smoothness on, 0 off
      priors.coeff_prior: boolean, 1 for deformation smoothness on, 0 off
Output
J: the Jacobian matrix pattern
```

We now run the code on the provided examples and present the output below:

- For the real experiment: **R**
  Initial reprojection error: 0.56616%
  Final reprojection error: 0.0031002%

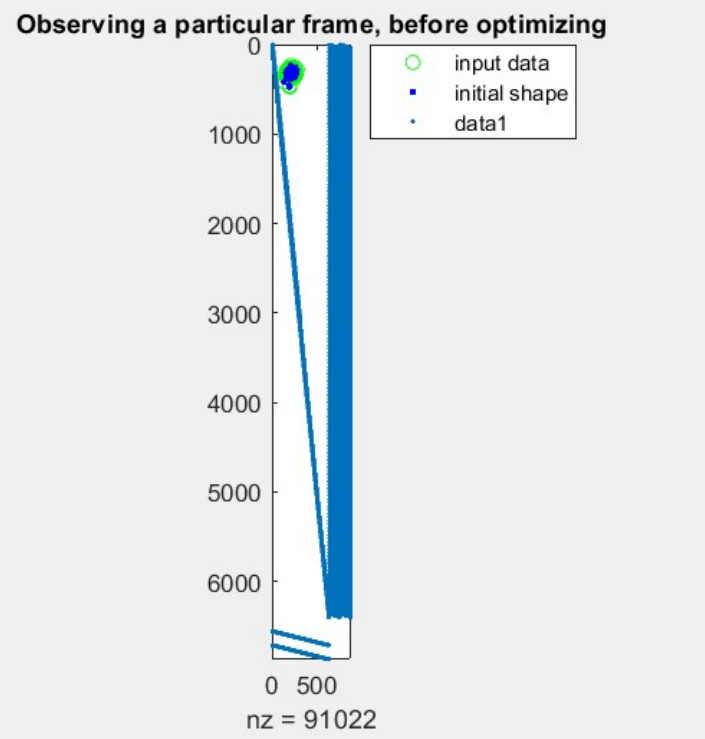


- For the synthetic experiment: **S**
  Initial reprojection error: 0.36214
  Final reprojection error: 0.0019342%
  3D error by assuming a rigid model: 3.1702 %
  3D error by assuming a non-rigid model: 2.503 %

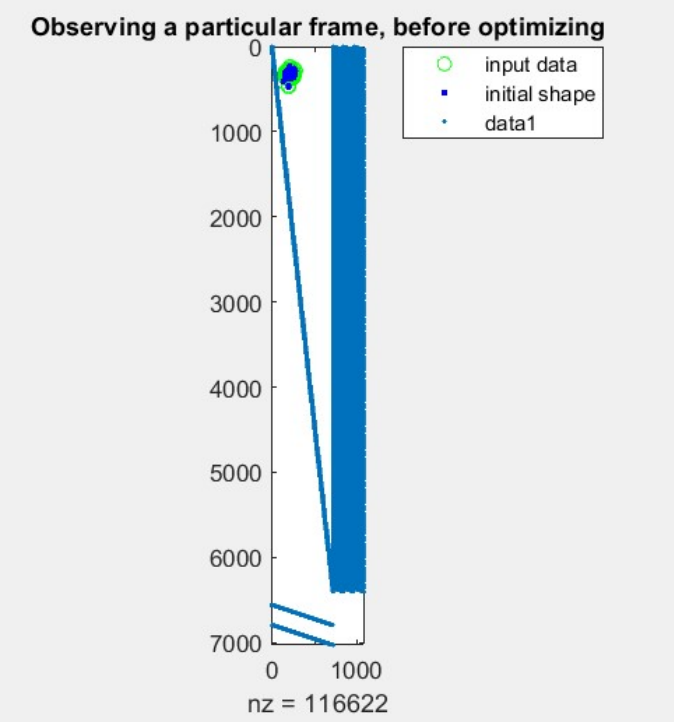


Now we run the synthetic case while adjusting the parameter K, while applying the temporal smoothness priors.
The resulting binary patterns with different parameter values are presented below:

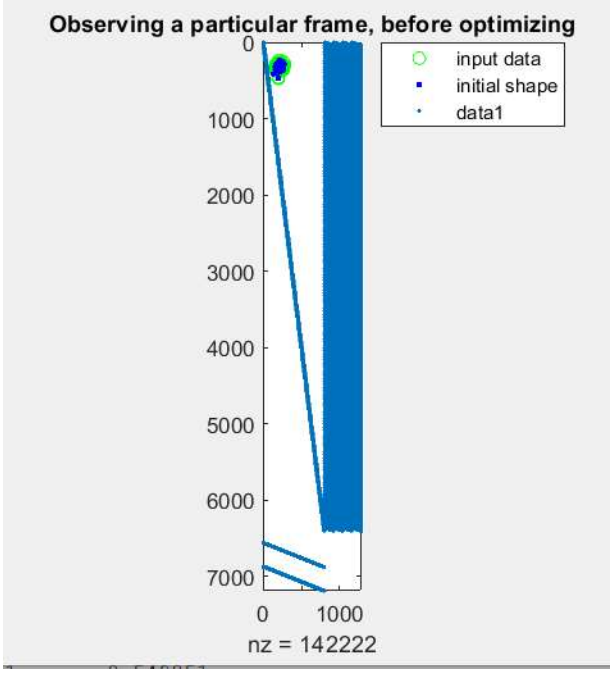
- K = 2


Observing a particular frame, before optimizing
input data
initial shape
data1
0
1000
2000
3000
4000
5000
6000
0
500
nz = 91022

- K = 3


Observing a particular frame, before optimizing
input data
initial shape
data1
0
1000
2000
3000
4000
5000
6000
7000
0
1000
nz = 116622

- K = 4


Observing a particular frame, before optimizing
input data
initial shape
data1
0
1000
2000
3000
4000
5000
6000
7000
0
1000
nz = 142222

u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva

1.4 Third Part:

Goal: Practice on the non-rigid structure from motion by assuming a trajectory linear subspace. We have a "orthographic camera", and factorization approach will be used. Here is some relevant functions:

- metricUpgrade.m: This file implements a function to solve the metric upgrade step in a trajectory subspace.
- recoverR.m: This file implements a function to obtain a matrix of size [2F x 3F] from the rotations per frame in a [2F x 3] matrix.
- DCT basis.m: This file implements a function to generate a DCT trajectory basis.

We have already implemented the function "nrsfm_trajectory.m" function. You can see the implemented function in the provided MATLAB file.

We adjusted our single value decomposition procedure from the first part for non-rigid shape reconstruction (see nrsfm_trajectory.m). Now the reconstruction depends on the rank of the trajectory basis. We run the code on various datasets provided and report the 3D errors (shape and rotation), depending on the rank. For each dataset, an optimal rank is highlighted in the table which is optimal for shape and rotation. You also can detect optimal ranks specific for shape and rotation values separately.

stretch

Rank (K)	3D error (shape)	3D error (rotation)
2	0.4112	0.096582
3	0.231	0.095322
4	0.99918	0.83226
5	0.21995	0.11668
6	1.416	1.042
7	0.14877	0.088982
8	0.17286	0.10582
9	0.15997	0.10248
10	0.12103	0.077162
11	0.84597	0.71296
12	**0.10878**	**0.054882**
13	0.11945	0.048904
14	-	-

u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva

drink

Rank (K)	3D error (shape)	3D error (rotation)
2	0.14841	0.030946
3	0.14387	0.022993
4	0.13974	0.011697
5	0.53438	0.49277
6	0.086745	0.010972
7	0.081993	0.007347
8	0.041733	0.006909
9	0.039902	0.0067641
10	0.038515	0.0061863
11	0.034452	0.0052503
12	0.028491	0.0054735
13	0.024957	0.0057904
14	-	-

yoga

Rank (K)	3D error (shape)	3D error (rotation)
2	0.29485	0.16505
3	0.24673	0.14835
4	0.24271	0.15037
5	0.6055	0.57307
6	0.50257	0.5789
7	0.20127	0.14981
8	0.16446	0.10777
9	1.2795	0.94873
10	0.34255	0.41792
11	0.16245	0.10592
12	0.24139	0.11306
13	0.33383	0.10502
14	-	-

u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva

pickup

Rank (K)	3D error (shape)	3D error (rotation)
2	0.4349	0.18076
3	0.35548	0.15106
4	0.54869	0.71665
5	0.30713	0.17192
6	0.28126	0.16952
7	0.25691	0.15758
8	0.24133	0.15414
9	0.26131	0.17039
10	0.25259	0.16368
11	0.16368	0.15743
12	0.23688	0.15492
13	0.27474	0.15465
14	-	-

dance

Rank (K)	3D error (shape)
2	0.57204
3	0.65187
4	0.4927
5	0.4927
6	0.29584
7	0.81484
8	0. 88168
9	0.88168
10	0.96619
11	0.77709
12	0.77523
13	0.89531
14	0.79139

u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva

dinosaur

Rank (K)	3D error (shape)
2	0.65874
3	0.65609
4	0.62391
5	0. 62658
6	0.59971
7	0.60761
8	0.60427
9	0.65713
10	0.63017
11	0. 62098
12	0.61943
13	0.62926
14	0.64165

u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva

1.5 Fourth Part:

To finish, you should consider the learning-based work **Neural Dense Non-Rigid Structure from Motion** with latent space constraints. Basically, you should briefly explain the main loss the authors are considering for training, and how the neural model is exploited to encode the deformation.

Answer:

In the paper "Neural Dense Non-Rigid Structure from Motion" which is proposed differently from classical formulations, the deformation models in an end-to-end unsupervised manner from 2D point tracks are regressed during training of neural network. The assumptions about the type of the detected deformations and the complexity can made through having an architecture and composition of the layers. This method is called "Neural Non-Rigid Structure from Motion". This deformation model is formulated by an auto-encoder model and imposed subspace constraints on the recovered latent space function in a frequency domain.

For the training of this model there are some of main loss functions are used. The non-rigid model as deformation autoencoder $f\theta$ is constructed as a series of nine fully connected layers with hidden dimensions which have exponential linear unit activations. The θ term represents learned network parameters. In addition, the autoencoder $f\theta$ is fully differentiable loss function and to learn θ, the author is used combination of fully differentiable and easily integrable into deep learning systems loss functions. Lastly E is defined as energy function which is compatible with autoencoder $f\theta$.

In order to solve the problem by minimizing a differentiable energy function E, the linear combination of several different energy functions Edata, is a data term, and {Etemp,Espat,Etraj,Elatent} with weight coefficients β, γ, η and ω are used. Here is the linear combination equation.

$$\mathbf{E} = \mathbf{E}_{\text{data}}(\boldsymbol{\theta}, \mathbf{z}, \mathbf{R}) + \beta\, \mathbf{E}_{\text{temp}}(\boldsymbol{\theta}, \mathbf{z}) + \gamma\, \mathbf{E}_{\text{spat}}(\boldsymbol{\theta}, \mathbf{z}) + \eta\, \mathbf{E}_{\text{traj}}(\boldsymbol{\theta}, \mathbf{z}) + \omega\, \mathbf{E}_{\text{latent}}(\mathbf{z}), \quad (4)$$

u195839 Umut Ekin Gezer
u204256 Ekaterina Erofeeva

For each energy function components there are different loss functions are used to minimize each energy function to solve the problem.

For the Edata component Huber loss of a matrix is used to penalize the image re-projection errors:

$$\mathbf{E}_{\text{data}}(\boldsymbol{\theta}, \mathbf{z}, \mathbf{R}) = \left\| \mathbf{W} - \mathbf{R}\left((\mathbf{1}_T \otimes \bar{\mathbf{S}}) + f_{\boldsymbol{\theta}}(\mathbf{z})\right) \right\|_{\epsilon}, \qquad (5)$$

The temporal smoothness term Etemp is minimized by enforcing through temporal-preserving regularization of the 3D shape via its latent space.

$$\mathbf{E}_{\text{temp}}(\boldsymbol{\theta}, \mathbf{z}) = \sum_{t=1}^{T-1} \left\| f_{\boldsymbol{\theta}}(\mathbf{z}_{t+1}) - f_{\boldsymbol{\theta}}(\mathbf{z}_t) \right\|_{\epsilon}. \qquad (6)$$

The spatial smoothness term Espat, which is responsible from spatial-preserving regularization for a neighborhood. The loss function of Espat is combination of two loss functions. First is Laplacian smoothing and second is depth control:

$$\mathbf{E}_{\text{spat}}(\boldsymbol{\theta}, \mathbf{z}) = \underbrace{\sum_{t=0}^{T-1} \sum_{\mathbf{p} \in \mathbf{S}_t} \left\| \mathbf{p} - \frac{1}{|\mathcal{N}(\mathbf{p})|} \sum_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} \mathbf{q} \right\|_1}_{\text{Laplacian smoothing}} - \lambda \underbrace{\sum_{t=1}^{T} \left\| \mathcal{P}_z(\mathbf{G}_t \mathbf{S}_t) \right\|_2}_{\text{depth control}}, \qquad (7)$$

Thanks to depth term with a weight coefficient $\lambda > 0$, the model achieves better supervision on the z-coordinate of the 3D shapes, which gives rise to increasing shape extent along the z-axis.

The energy function Etraj is trained through the penalty term based on the 3D point trajectories are coded by linear combination of K fixed trajectory vectors by a T × K matrix Φ together with a 3K × P matrix A of unknown coefficients. Here is the loss function of Etraj.

$$\mathbf{E}_{\text{traj}}(\boldsymbol{\theta}, \mathbf{z}) = \left\| (\mathbf{1}_T \otimes \bar{\mathbf{S}}) + f_{\boldsymbol{\theta}}(\mathbf{z}) - (\boldsymbol{\Phi} \otimes \mathbf{I}_3)\mathbf{A} \right\|_{\epsilon}, \quad \boldsymbol{\Phi} = \begin{pmatrix} \phi_{1,1} & \cdots & \phi_{1,K} \\ \vdots & \ddots & \vdots \\ \phi_{T,1} & \cdots & \phi_{T,K} \end{pmatrix}, \quad (8)$$

Finally, the loss function, with Fourier transform operator which solves sparsity constraints by exploiting sparsity of Fourier Series, is used. Here is the last loss function of Elatent.

$$\mathbf{E}_{\text{latent}}(\mathbf{z}) = \left\| \mathcal{F}(\mathbf{z}) \right\|_1, \quad (9)$$