Flow supervision for Deformable NeRF Supplementary Material

A. Implementation details

A.1. SE(3) transformation in normalized device coordinate

For unbounded frontal scenes in the DVS dataset [7], normalized device coordinate (NDC) [3] is required to squeeze unbounded 3D space into a bounded one with respect to the depth direction. To perform SE(3) transformation of points with the NDC coordinates, naive implementation would be first convert the NDC coordinates to Euclidean coordinates, apply the SE(3) transformation, and then convert it back to the NDC coordinates. However doing so may run into numerical instability issue for points in long distance. Alternatively we derived a formula to directly apply SE(3) transformation in the NDC coordinates. For any NDC coordinates (x, y, z), its corresponding NDC coordinates (x', y', z') after applying the SE(3) transformation with rotation $\mathbf{R} \in SO(3)$ and translation $\mathbf{t} \in \mathbb{R}^3$ is calculated as follow:

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \mathbf{R} \begin{pmatrix} \frac{2f_x x}{W} \\ \frac{2f_y y}{H} \\ 1 \end{pmatrix} + \frac{1-z}{2n}\mathbf{t} \tag{1}$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \frac{2f_x a}{Wc} \\ \frac{2f_y b}{Hc} \\ \frac{1+c-z}{c} \end{pmatrix}, \tag{2}$$

where f_x, f_y, H, W, n are the focal lengths, image sizes and near plane distance as in the definition of NDC coordinates. The above formulation avoids the hassle of dealing with points at infinity in the Euclidean coordinates.

A.2. Efficient implementation of equation (5)

To be able to backpropagate through equation (5) and make the computation tractable for evaluating on 100k+ points per iteration, we made the following implementation design choices.

First, we choose to implement a 2nd-order differentiable quaternion-based operator using CUDA. This is 100x faster than implementation with native pytorch operators as in PyTorch3D [5], and 5x faster than matrix multiplication based implementation. We note that LieTorch library [6] also provides fast implementation of SE(3) transformation and computes gradients in the tangent space. However it does not support 2nd-order derivatives which is required for optimization with respect to the velocities estimated by equation (5).

For calculating the 3×3 matrix inverse of the Jacobian matrices, me choose to implement an analytical solution of the matrix inverse with CUDA, which is 100x faster than pytorch's generic matrix inverse routine when dealing with 100k 3×3 small matrices. We use the following equation to calculate the derivative,

$$\frac{\partial \mathbf{H}^{-1}}{\partial h_{ij}} = -\mathbf{H}^{-1}\boldsymbol{\delta}_{ij}\mathbf{H}^{-1}, \tag{3}$$

where $\boldsymbol{\delta}_{ij}$ is a binary matrix whose (i, j)th entry is the element with value of 1. In our CUDA implementation, we parallalize the compute for each element of $\frac{\partial \mathbf{H}^{-1}}{\partial h_{ij}}$ so as to be significantly faster than the generic matrix inverse operators from pytorch.

B. Comparison to directly inverting the backward deformation field.

In Fig. 1, we compare our method against two baselines which directly invert the backward deformation field. The experiment setup is to fit a video with se(2) motions as described in Figure 4 of the main paper. The optimization objective is to minimize the combination of photometric loss and optical flow loss.

The first baseline (3rd column) has a separate MLP which represents the *forward* deformation field $w_{c\rightarrow}$. It has the same architecture as the *backward* deformation field $w_{c\leftarrow}$. Then the optical flow between frame t and $t + \Delta t$ at pixel $\mathbf{p}$ is estimated as $\mathbf{o}_{t\rightarrow t+\Delta t} = w_{c\rightarrow}(w_{c\leftarrow}(\mathbf{p}, t), t + \Delta t) - \mathbf{p}$. As shown in the 3rd column of the following figure, this baseline is not able to reconstruct the input video with high fidelity (PSNR=24.8). This is due to there being no explicit gaurantee that the forward and backward deformation fields are cyclic consistent.

The 2nd baseline applies normalizing flow as in Lei & Daniilidis [1] The deformation field is modeled by RealNVP, which is a bijective mapping with analytic inverse.

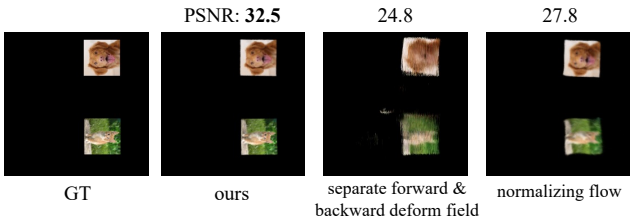

Figure 1. Compare flow supervision using different deformation representation. Our method (2nd column) outperforms baselines using separate forward & backward deformation field (3rd column) and bijective normalizing flow (4th column).

We find that due to the network architecture restrictions, Real-NVP is not able to perfectly fit the motions (see distortion of image patches in the 4th column), even when it has significantly more layers than our method (12 vs 4).

In comparison, our method achieves the highest PSNR which indicates its effectiveness against directly inverting the backward deformation field. Finally, we note that it may be possible to make the baselines stronger by carefully tuning loss functions or network architectures, however the main strength of our method is its generality without the need for introducing additional modules or tuning.

C. Other optical flow rendering/loss alternatives

In our preliminary investigation, we also experimented different ways of rendering the optical flow from 3D scene flows and evaluating the flow loss. All of the following alternatives are inferior to the one we described in the main paper, and we have recorded them here for reference.

- The first alternative is instead of evaluating scene flows for all sampled points along the ray and then weighted averaging them, we only evaluate scene flows for a single point on the estimated visible surface. More specifically, we first synthesize the depth as in NeRFs [2–4] by weighted averaging depth of sampled points along the ray. This gives an estimation of a position $\mathbf{p}(t)$ on the visible surface. Then the next position $\mathbf{p}(t + \Delta t)$ is estimated by equation (7).

- We still evaluate scene flows for all the points along the ray. But instead of aggregate them together to form a single optical flow, we project all scene flows to 2D flows, and directly evaluate optical flow loss by comparing them to the input optical flow. To account for occlusions, we weighted the loss by the weights from the volumetric rendering equation. More specifically, given the projected 2D flows $\mathbf{o}_i \in \mathbb{R}^2$ for each point i along the ray, we evaluate the optical flow loss as follow:

$$\mathcal{L}_{\text{o.f.}} = \sum_i w_i \|\mathbf{o}_i - \mathbf{o}_{\text{input}}\|_1, \tag{4}$$

  where w_i denotes the weights from the volumetric rendering and $\mathbf{o}_{\text{input}} \in \mathbb{R}^2$ is the input optical flow.

In our current experiment, we found both above approaches are inferior to the one we used in the main paper. Sometimes they resulted in training instability or floating surfaces. A deeper understanding of why these two approaches do not work may motivate more efficient approaches of enforcing the flow loss.

References

[1] Jiahui Lei and Kostas Daniilidis. Cadex: Learning canonical deformation coordinate space for dynamic surface representation via neural homeomorphism. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6624–6634, 2022. 1

[2] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. *arXiv preprint arXiv:2011.13084*, 2020. 2

[3] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2

[4] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948*, 2020. 2

[5] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 1

[6] Zachary Teed and Jia Deng. Tangent space backpropagation for 3d transformation groups. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 1

[7] Jae Shin Yoon, Kihwan Kim, Orazio Gallo, Hyun Soo Park, and Jan Kautz. Novel view synthesis of dynamic scenes with globally coherent depths from a monocular camera. In *CVPR*, 2020. 1