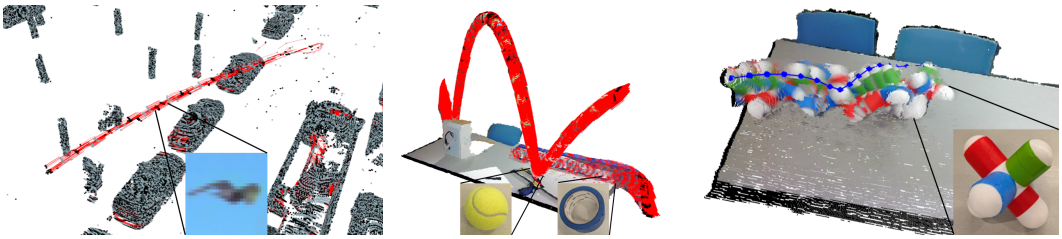
Neural Eulerian Scene Flow Fields

Kyle Vedder[1,2*] **Neehar Peri**[2,3] **Ishan Khatri**[3] **Siyi Li**[1] **Eric Eaton**[1]
Mehmet Kocamaz[2] **Yue Wang**[2] **Zhiding Yu**[2] **Deva Ramanan**[3] **Joachim Pehserl**[2]
[1]University of Pennsylvania [2]NVIDIA [3]Carnegie Mellon University



(a) Small object motion extraction... (b) ...in diverse, dynamic scenes... (c) ...with emergent 3D point tracking behavior!

Figure 1: EulerFlow is able to capture the motion of small, fast moving objects with few lidar points, such a bird flying in front of an autonomous vehicle (Figure 1a). EulerFlow's flexibility allows it to estimate scene flow for fast-moving table top objects *without additional hyperparameter tuning* (Figure 1b). EulerFlow's PDE estimate exhibits emergent 3D point tracking behavior without explicit long-horizon supervision (Figure 1c). Note that point clouds are shown in color for visualization purposes only; RGB is not used during optimization.

Interactive scene visualizations at `vedder.io/eulerflow`

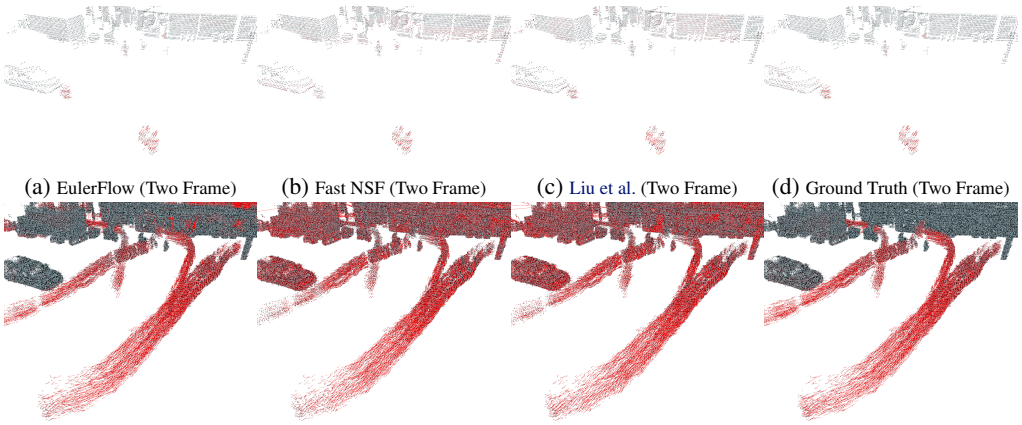
Abstract: We reframe scene flow as the task of estimating a continuous space-time PDE that describes motion for an entire observation sequence, represented with a neural prior. Our method, *EulerFlow*, optimizes this neural prior estimate against several multi-observation reconstruction objectives, enabling high quality scene flow estimation via pure self-supervision on real-world data. EulerFlow works out-of-the-box without tuning across multiple domains, including large-scale autonomous driving scenes and dynamic tabletop settings. Remarkably, EulerFlow produces high quality flow estimates on small, fast moving objects like birds and tennis balls, and exhibits emergent 3D point tracking behavior by solving its estimated PDE over long-time horizons. On the Argoverse 2 2024 Scene Flow Challenge, EulerFlow outperforms *all* prior art, surpassing the next-best *unsupervised* method by more than 2.5×, and even exceeding the next-best *supervised* method by over 10%.

1 Introduction

Scene flow estimation is the task of describing motion with per-point 3D motion vectors between temporally successive point clouds [1, 2, 3, 4, 5, 6, 7]. Such per-point motion estimates are critical for autonomy in diverse environments, e.g., maneuvering around open-world objects like debris [8] or folding deformable cloth [9]. Importantly, scene flow estimation requires not only an understanding of object *geometry*, but also its *motion*. However, scene flow methods broadly do not work on smaller objects [7]. For example, in the autonomous vehicles domain, Khatri et al. highlight that even supervised methods struggle to describe the majority of pedestrian motion, with unsupervised methods failing dramatically. Scene flow promises to be a powerful primitive for understanding the dynamic world, but such failures explain why it has limited adoption in downstream applications like tracking [10] or open-world object extraction [11].

*Corresponding email: `kvedder@seas.upenn.edu`

Workshop on Safe and Robust Robot Learning for Operation in the Real World (SAFE-ROL) at CoRL 2024.



(a) EulerFlow (Two Frame) (b) Fast NSF (Two Frame) (c) Liu et al. (Two Frame) (d) Ground Truth (Two Frame)

(e) EulerFlow (Full Sequence) (f) Fast NSF (Full Sequence) (g) Liu et al. (Full Sequence) (h) Ground Truth (Full Sequence)

Figure 2: We visualize an example of five pedestrians crossing the street in front of a stopped car, cherrypicked to have unusually high density lidar returns, making it particularly easy to estimate flow. Figures 2a–2d depict a two-frame flow visualization of EulerFlow and several strong baselines. Notably, only visualizing flow over two frames makes it difficult to distinguish flow quality. In contrast, Figures 2e–2h depict flow vectors over the full sequence, making differences in quality clear; for example, EulerFlow is the only one without artifacts on the stopped car.

Scene Flow via ODE. In Figure 2, visual assessment of scene flow quality is much easier in an accumulated global frame; while incomplete due to an implicit time axis, these accumulated flow vectors allow viewers to imagine how positions in 3D space evolve over *many* timesteps, and compare that to predicted flows. This imagination of scene flow as continuous motion over large time intervals motivates us to model scene flow as an ordinary differential equation (ODE) that describes the scene's instantaneous motion across continuous position and time. Scene flow estimation then becomes the task of estimating this ODE. We can straightfowardly represent this ODE estimate with a neural prior [13] and optimize it against scene flow surrogate objectives, both over single frame pairs and extended *across arbitrary time intervals* to produce better quality estimates. We formalize this in Section 3 and propose the *Scene Flow via ODE* framework.

EulerFlow. We instantiate Scene Flow via ODE with standard point cloud distance objectives like Chamfer Distance to create *EulerFlow*. Notably, EulerFlow outperforms *all* prior art, supervised or unsupervised, on the Argoverse 2 2024 Scene Flow Challenge and Waymo Open Scene Flow benchmark. It outperforms prior *unsupervised* methods by a large margin ($> 2.5\times$ mean dynamic error reduction), and is able to capture small, fast moving objects, including those outside of labeled taxonomies (e.g. the flying bird in Figure 1a). Due to its simplicity, EulerFlow is able to provide high quality scene flow out-of-the-box on real-world data for other important domains such as dynamic tabletop settings (Figure 1b) *without* domain-specific tuning. Finally, we show that simple ODE solving techniques like Euler integration can be used to extract 3D point tracks (Figure 1c), which serves as both an exciting emergent behavior as well as a mechanism for visualizing and interpreting the quality of the continuous ODE estimate.

We present four primary contributions:

- We propose *Scene Flow via ODE* (SFvODE), a reframing of scene flow estimation as the task of fitting an ODE that describes the change of continuous positions over continuous time, unlocking a new class of surrogate objectives that enable better scene flow estimates.
- We instantiate SFvODE with *EulerFlow*, a flexible **unsupervised** scene flow method that achieves **state-of-the-art** performance on the Argoverse 2 2024 Scene Flow Challenge, **beating all prior supervised and unsupervised methods**.
- We study EulerFlow and show its strong performance is derived from its ability to optimize its ODE estimate against the full sequence of observations over arbitrary time horizons.
- We show that EulerFlow's simple, flexible formulation allows it to run unmodified on a variety of domains, with emergent capabilities like 3D point tracking behavior.

2 Background and Related Work

Evaluation. Dewan et al. formalized scene flow for point clouds as the task of estimating motion between point cloud P_t at time t and point cloud P_{t+1} at $t+1$ by estimating the true flow $\mathcal{F}_{t,t+1}$, i.e. true residual vectors for every point in P_t that describe its movement to its associated position at $t+1$. Error is computed by measuring the per-point endpoint distance between estimated and ground truth vectors. Historically, these errors are reported with a per-point average (*Average EPE*); however, as Chodosh et al. show, Average EPE is dominated by background points, preventing meaningful measurement of non-ego object motion descriptions. Khatri et al. address this shortcoming with *Bucket Normalized EPE*, which reports per-class performance normalized by speed, allowing for direct comparisons across classes with very different average speeds (e.g. pedestrians and cars). Bucket Normalized EPE is the basis for the *Argoverse 2024 Scene Flow Challenge*[1], where methods are ranked by the mean error of their motion descriptions (*mean Dynamic Normalized EPE*).

Input / Output Formulation. Dewan et al.'s choice to formulate scene flow using *only* two input frames is arbitrary; it's the minimal information needed to extract rigid motion, but there are not real-world problems constrained to *only* have access to two frames. Indeed, using five or ten frames of past observations is standard practice in the 3D detection literature [15, 16, 17, 18, 19], and multi-frame formulations have started to appear in the scene flow literature: Liu et al. [12] and Flow4D [20] use three (P_{t-1}, P_t, P_{t+1}) and five input frames $(P_{t-3}, \dots, P_{t+1})$ respectively to predict $\hat{\mathcal{F}}_{t,t+1}$. As we discuss in Section 3, rather than just using more observations to estimate flow for a single frame pair, we formulate scene flow as a joint estimation problem: given the full observation sequence $(P_0, \dots, P_N)$, we estimate *all* flows $\hat{\mathcal{F}}_{0,1}, \dots, \hat{\mathcal{F}}_{N-1,N}$ between *all* adjacent observations.

Feedforward Methods. Feedforward networks are a popular class of scene flow methods due to their fast inference speed [2, 21, 22, 23, 24, 25, 26, 4, 27, 28, 29, 20, 30]. While they are often trained with supervised labels, recent work has developed distillation pipelines that leverage unsupervised pseudolabelers [6, 5, 31].

Neural Scene Flow Prior. Li et al. [13] propose Neural Scene Flow Prior (NSFP), a widely adopted unsupervised scene flow approach. NSFP uses the inductive bias of the smooth, restricted learnable function class of two ReLU MLP coordinate networks (8 hidden layers of 128 neurons); θ to estimate forward flow and θ' to estimate backwards flow, minimizing

$$\text{TruncatedChamfer}(P_t + \theta(P_t), P_{t+1}) + \left\| P_t + \theta(P_t) + \theta'(P_t + \theta(P_t)) - P_t \right\|_2 , \tag{1}$$

where TruncatedChamfer is defined as the standard L_2 Chamfer distance, but with per-point distances above 2 meters set to zero in order to reduce the influence of outliers. NSFP is optimized for at most 1000 steps with early stopping.

Motion Beyond Two Frames. Wang et al. [32] tackles the adjacent problem of estimating 3D point *trajectories* over 25 frames with Neural Trajectory Prior (NTP) by jointly optimizing three separate ReLU MLP neural priors: 1) a sinusoidal embedded, time conditioned, 25 frame trajectory basis estimator (embed$(t) \mapsto 256 \times 25 \times 3$ tensor, where 256 is the dimension of the trajectory basis), 2) a coordinate network bottleneck encoder, and 3) a bottleneck decoder to estimate a per-point linear combination over the learned trajectories. Trajectories are optimized for both a one-frame lookahead L_2 Chamfer loss and a cyclic consistency loss over the entire velocity space trajectory.

Deformation in Reconstruction. Nerfies [33] and DynamicFusion [34] estimate a deformation field to warp a canonical frame to explain the observed frame. While capable of describing small motions, these methods require a canonical frame that contains all of the relevant geometry to deform; however, in large, highly dynamic scenes like autonomous driving, there is often no frame that contains all moving objects. By comparison, Scene Flow via ODE does not assume the existence of a canonical frame, instead only describing how the scene changes.

[1] https://www.argoverse.org/sceneflow

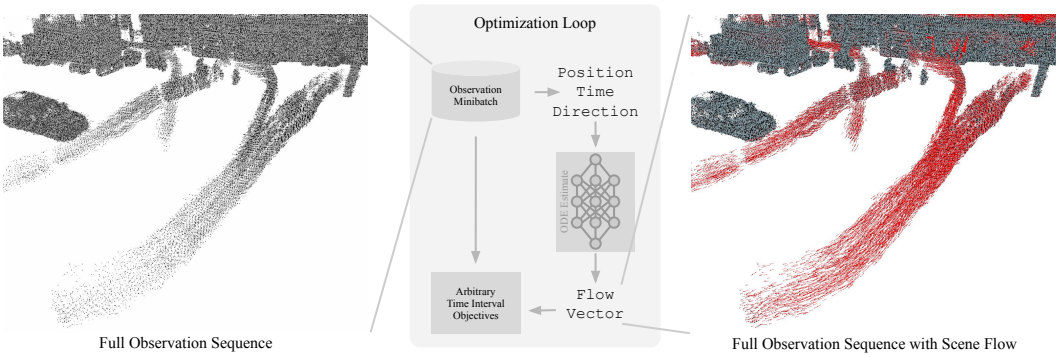


Figure 3: Overview of our *Scene Flow via ODE* framework, which estimates an ODE across the entire observation sequence by optimizing against multi-frame objectives. This ODE estimate is backed by a neural prior [13], providing a general representation for describing position-time motion.

3 Scene Flow via ODE

Prior art consumes multiple frames $(P_{t-N}, \ldots, P_{t+1})$ as input, but these methods are ultimately only tasked with estimating flow vectors between P_t and P_{t+1}. We instead pose the problem of estimating a time-conditioned flow field that describes motion for *all* adjacent point clouds P_t, P_{t+1} in the entire sequence $(P_0, \ldots, P_N)$. To do this, rather than describing scene flow as positional change over a fixed interval ($\mathcal{F}_{t,t+1}$ are residual vectors over the interval t to $t+1$) as we did in Section 2, we can instead express these changes as a differential equation that describes positional change over *continuous* time.

Formally, given a scene, let $L(x_0, y_0, z_0, t)$ be the Lagrangian view of the scene's true flow field, i.e. a continuous function that, based on a canonical frame at time 0, describes the true position of the canonical frame particle x_0, y_0, z_0 at some other time t. As we discuss in Section 2, this Lagrangian view is common in the the deformable reconstruction literature, and the requirement for a canonical frame definition means these approaches struggle to describe scenes where there is no frame that contains all moving objects.

To break this canonical frame dependence, we choose to take an Eulerian view of the flow field, i.e. $F = \frac{dL}{dt}$, which describes the velocity of a query point at some arbitrary time. As we show in our derivation in Appendix E, this formulation does not require point correspondences in some other canonical frame when estimating a point's trajectory from t to t'; instead, we can simply set the initial conditions of the ODE at t to x_t, y_t, z_t and utilize an off-the-shelf ODE solver (e.g. Euler integration) to extract flow from t to t', expressed as $E(x_t, y_t, z_t, t, t')$.

We do not know the true flow field F when estimating scene flow; however, we can represent F with a neural prior θ ($F \approx \theta$), and optimize θ against surrogate objectives. This framing, which we formalize into the *Scene Flow via ODE* framework (SFvODE; Figure 3), allows θ to benefit from constructive interference between objectives, as well as enables us to formulate objectives over arbitrarily long time horizons, unlocking high quality estimates.

4 EulerFlow

Scene Flow via ODE proposes a framework where the neural prior θ represents an estimate of the Eulerian flow field F (i.e. $F \approx \theta$); however, it does not prescribe the optimization objectives for θ. Thus, we instantiate Scene Flow via ODE with *EulerFlow*, a point cloud only scene flow method[2] with reconstruction and cyclic consistency objectives across the entire sequence of observations.

As we show in Equation 17 (Appendix E.4), we can use θ's Eulerian flow field estimate to extract an estimated point trajectory from x_t, y_t, z_t at t to some future location at time t' via Euler integration

[2]Visualizations shown in color for better viewing. EulerFlow can also use monodepth estimates (Appendix B.2)

over θ without requiring a canonical frame definition, i.e. $E_\theta(x_t, y_t, z_t, t, t')$. By extracting point trajectories for every point p in P_t using E_θ, we can not only construct a two-frame scene flow estimate of $\mathcal{F}_{t,t+1}$, but also estimate flow to arbitrary future or prior timesteps (e.g. $\mathcal{F}_{t,t+2}$ or $\mathcal{F}_{t,t-1}$). This allows us to optimize over multi-frame reconstruction objectives: we can now pose reconstruction surrogate objectives between *any* two point clouds in our observation sequence, not just adjacent point clouds P_t and P_{t+1}. Similarly, we can straightforwardly pose cyclic consistency objectives by composing $\mathcal{F}_{t,t+1}$ and $\mathcal{F}_{t+1,t}$. Formally, for P_t's $\mathcal{F}_{t,t+k}$ (for any $k \in \mathbb{Z}$), we define

$$\text{Euler}_\theta\left(P_t, k\right) = P_t + \mathcal{F}_{t,t+k} = \forall p \in P_t : E_\theta(p_{xt}, p_{yt}, p_{zt}, t, t+k) \ , \tag{2}$$

enabling us to pose θ's optimization objective $\forall P_t \in (P_0, \ldots, P_N)$ across the window of size W

$$\arg\min_\theta \sum \begin{array}{l} \forall k \in \{-W, \ldots, W\} \setminus \{0\} : \text{TruncatedChamfer}(\text{Euler}_\theta\left(P_t, k\right), P_{t+k}) \\ \alpha \left\|\text{Euler}_\theta\left(\text{Euler}_\theta\left(P_t, 1\right), -1\right) - P_t\right\|_2 \end{array} \tag{3}$$

In practice, we set W to 3 and α to 0.01. We provide additional implementation details in Appendix D. In order to optimize θ, our estimate of the Eulerian flow field F, we perform Euler integration to extract point cloud flow estimates as part of reconstruction losses. Notably, EulerFlow only requires a single optimization loop over a single neural prior θ compared to NSFP's two priors θ and θ'. Our neural prior θ is a straightforward extension to NSFP's coordinate network prior. Like with NSFP, TruncatedChamfer is defined as the standard L_2 Chamfer distance with per-point distances below 2 meters. As we show in Section 5, EulerFlow's simple ODE estimation formulation across multiple observations produces high quality flow, and solving this ODE over arbitrary time spans unlocks emergent point tracking behavior.

5 Experiments

In order to validate EulerFlow's construction and better understand the impact of its design choices, we perform extensive experiments on the Argoverse 2 [35] and Waymo Open [36] autonomous vehicle datasets. We compare against open source implementations of FastNSF [37], Liu et al., NSFP [13], FastFlow3D [4], and variants of ZeroFlow [6] provided by the ZeroFlow model zoo[3], a third-party implementation of NTP [32] from Vidanapathirana et al., and Argoverse 2 2024 Scene Flow Challenge leaderboard submission results from the authors of Flow4D [20], TrackFlow [7], DeFlow++/DeFlow [30], ICP Flow [31], and SeFlow [5]. As discussed in Khatri et al. and used in the Argoverse 2 2024 Scene Flow Challenge, methods are ranked by their speed normalized *mean Dynamic Normalized EPE*.

Implementation Details. To showcase the flexibility of EulerFlow without hyperparameter tuning, for all quantitative experiments we run with a neural prior of depth 8 (NSFP's default depth), except for our submission to the Argoverse 2 2024 Scene Flow Challenge (Section 5.1) where, based on our depth ablation study on the val split (Section 5.2.3), we set the depth of the neural prior to 18. As discussed in NTP's original paper [32] and confirmed by our experiments, NTP struggles to converge beyond 25 frames, so we only compare against it in a 20 frame settings. As is typical in the scene flow literature [14], we perform ego compensation and ground point removal on both Argoverse 2 and Waymo Open using the dataset provided map and ego pose.

5.1 How does EulerFlow compare to prior art on real data?

EulerFlow achieves **state-of-the-art** performance on the *Argoverse 2 2024 Scene Flow Challenge* leaderboard. Despite being unsupervised, EulerFlow **surpasses *all* prior art, supervised or unsupervised**, including Flow4D [20][4] and ICP Flow [31][5]. Notably, EulerFlow achieves $< 2.5\times$ lower error mean Dynamic EPE than ICP Flow and beats Flow4D by over 10%.

[3] https://github.com/kylevedder/SceneFlowZoo, from Vedder et al. [6].

[4] Flow4D is the winner of the 2024 Argoverse 2 Scene Flow Challenge supervised track.

[5] ICP Flow is the winner of the 2024 Argoverse 2 Scene Flow Challenge unsupervised track.

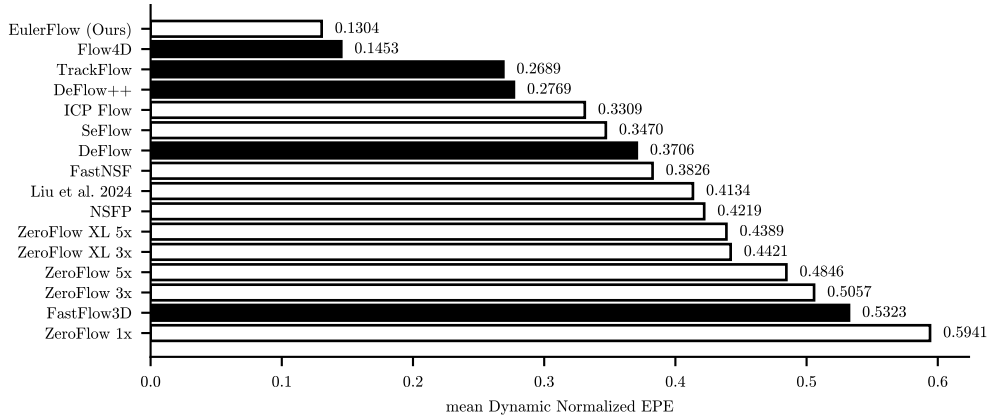


Figure 4: Mean Dynamic Normalized EPE of EulerFlow compared to prior art on the Argoverse 2 2024 Scene Flow Challenge test set. EulerFlow is state-of-the-art, beating all supervised (shown in black) and unsupervised (shown in white) methods. Lower is better.

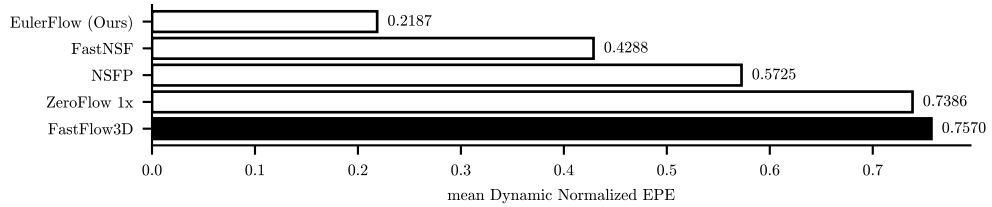


Figure 5: Mean Dynamic Normalized EPE of EulerFlow compared to prior art on the Waymo Open validation set. EulerFlow is state-of-the-art, beating all supervised (shown in black) and unsupervised (shown in white) methods. Lower is better.

EulerFlow's dominant performance also holds on Waymo Open [36]; we compare against several popular methods (Figure 5), and EulerFlow again out-performs the baselines by a wide margin, more than halving the error over the next best method.

5.2 What contributes to EulerFlow's state-of-the-art performance?

We find that EulerFlow's lower mean Dynamic EPE can be attributed to better performance on smaller objects. On Argoverse 2, compared to Flow4D, EulerFlow's improves on `WHEELED VRU` (Figure 6d), a small, rare, fast moving class. Compared to ICP Flow, EulerFlow's improves on all classes (at least halving the error on every class!), with the largest improvements coming from the smaller and harder to detect objects `PEDESTRAIN` and `WHEELED VRU` (Figures 6c–6d). On Waymo Open, the same story holds; the most dramatic performance improvements come from the small object classes of `CYCLIST` and `PEDESTRIAN` (Figure 7).

These results are consistent with our qualitative visualizations. Figure 12 shows EulerFlow is able to cleanly extract the motion of a bird flying past the ego vehicle. Euler integration using EulerFlow's ODE, starting at the bird's takeoff position and ending when it loses lidar returns, produces emergent 3D point tracking behavior on the bird through its trajectory (Figure 8), further demonstrating the quality of EulerFlow's model of the scene's motion.

5.2.1 How does observation sequence length impact EulerFlow?

As we discuss in Section 3, EulerFlow benefits from constructive interference from ODE estimation over many observations. Does this sufficiently explain EulerFlow's performance? Figure 9 shows the performance of EulerFlow at length 5, 20, 50, and full sequence (roughly 160 frames) compared to NSFP and NTP at length 20. EulerFlow sees clear continual improvements as the number of frames increases without signs of saturation. However, sequence length alone does not explain EulerFlow's performance; even at the same sequence length of 20, EulerFlow demonstrates significantly better performance than NTP.

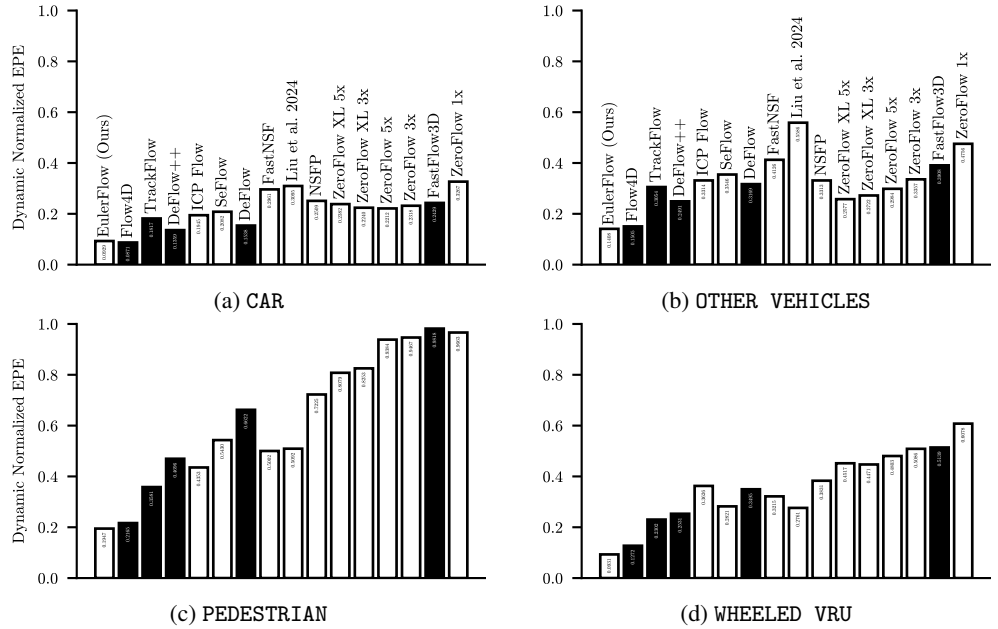


Figure 6: Per class Dynamic Normalized EPE of EulerFlow compared to prior art on the Argoverse 2 2024 Scene Flow Challenge test set. Supervised methods shown in black, unsupervised methods shown in white. Methods are ordered left to right by increasing mean Dynamic Normalized EPE. Lower is better.

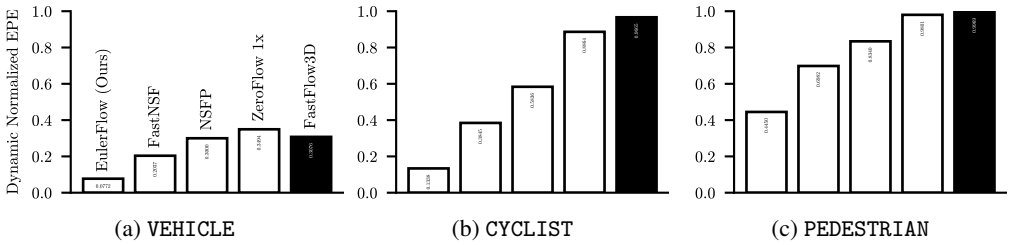


Figure 7: Per class Dynamic Normalized EPE of EulerFlow compared to prior art on the Waymo Open validation set. Supervised methods shown in black, unsupervised methods shown in white. Methods are ordered left to right by increasing mean Dynamic Normalized EPE. Lower is better.

5.2.2 How do multi-frame optimization objectives impact EulerFlow?

Equation 3 outlines two major components of EulerFlow's loss: multi-frame Euler integration for Chamfer Distance reconstruction, and cycle consistency. Figure 10 compares EulerFlow without more than one integration step (No $k > 1$) and without cycle consistency rollouts (No Cycle) to better understand the impact of these components. Ablating multi-step Euler integrated rollouts results in significant degredation, as they are a strong forcing function to have consistent, smooth flow volumes; indeed, despite consuming the entire sequence, EulerFlow (No $k > 1$) is only slightly better than NTP with a sequence length of 20. These results highlight the power of multi-step rollouts and their potential as a objective for other test-time optimization methods and feedforward methods.

5.2.3 How does the capacity of the neural prior impact EulerFlow?

Li et al. ablate the capacity of NSFP's neural prior to characterize underfitting and overfitting to optimization objective noise, ultimately selecting a depth of 8. EulerFlow's neural prior is structured similarly; however, NSFP is fitting a single snapshot in time, while EulerFlow is fitting an entire ODE over significant time intervals. Intuitively, one would expect that full sequence modeling would benefit from greater network capacity.

To evaluate this, we perform a sweep of EulerFlow's network depth on the Argoverse 2 validation split (Figure 11). While EulerFlow with NSFP's default of depth 8 performs well on our Argoverse 2 evaluations (0.1% worse than the supervised state-of-the-art Flow4D), we see that performance improves as the neural prior's depth increases until depth 18 (indicating underfitting), where we start to see degradation (indicating overfitting to noise). Based on these results our Argoverse 2 2024 Scene Flow Challenge leaderboard submission uses a depth 18 neural prior (Figure 4).

5.3 Beyond Autonomous Vehicles

Due to a dearth of real-world, labeled scene flow data, prior scene flow work on real data overwhelmingly evaluates on autonomous vehicle datasets [1, 13, 4, 37, 14, 12, 6, 7]; consequently, motion understanding in other important domains like tabletop manipulation has been neglected. To showcase EulerFlow's out-of-the-box flexibility and generalizability, we visualize EulerFlow on several dynamic tabletop scenes we collected using the ORBBEC Astra, a low cost depth camera commonly used in robotics (Figure 13). For viewing ease, we paint our point clouds with color; however, RGB information is not provided to EulerFlow during optimization. While EulerFlow only reasons about point clouds, it can leverage video mono depth estimates to describe RGB-only scene flow (Appendix B.2). Interactive visuals are available at `vedder.io/eulerflow`.

6 Conclusion

By reframing scene flow as fitting an ODE over positions for a full sequence of observations, we are able to construct EulerFlow, a simple unsupervised scene flow method that achieves state-of-the-art performance on the Argoverse 2 2024 Scene Flow Challenge and Waymo Scene Flow benchmark, where it beats all prior art, supervised or unsupervised. EulerFlow is able to describe motion on small, fast moving, out of distribution objects unable to be captured by prior art, suggesting that it makes good on the promises of scene flow as a powerful primitive for understanding the dynamic world. It also exhibits other emergent capabilities, like basic 3D point tracking behavior.

We believe that this ODE formulation has implications for scene flow at large, including beyond test-time optimization methods; the power of multi-step Euler integration may translate to feedforward network training. Future work should explore feedforward models that perform autoregressive rollouts or directly learn to estimate multiple steps into the future.

6.1 Limitations and Future Work

EulerFlow's strong performance opens the book on an exciting new line of work; however, we feel that it's important to be candid about EulerFlow's current limitations in order to make future progress.

EulerFlow is point cloud only. Point cloud sparsity bottlenecks performance; for instance, in Figure 8 and Figure 12 we were only able to track the bird for 20 frames because we lost lidar observations of the bird, while it remained visible in the car's RGB cameras. Future works should explore multi-modal fusion for better long-term motion descriptions.

EulerFlow is expensive to optimize. With our implementation, optimizing EulerFlow for a single Argoverse 2 sequence takes 24 hours on one NVIDIA V100 16GB GPU, putting it on par with the original NeRF paper's computation expense [39]. However, like with NeRF, we believe algorithmic, optimization, and engineering improvements can significantly reduce runtime.

EulerFlow does not understand ray casting geometry. During ego-motion, a static foreground occluding object casts a moving shadow on the background; this causes Chamfer Distance to estimate this as a leading edge of moving structure, encouraging false motion artifacts [13]. This can be addressed with optimization losses that model point clouds as originating from a time of flight sensor with limited visibility, as has been successfully demonstrated in the reconstruction [40] and forecasting literature [41, 42], rather than an unstructured set of points to be associated via local point distance.

References

[1] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard. Rigid scene flow for 3d lidar scans. In *Int. Conf. Intel. Rob. Sys.*, pages 1765–1770. IEEE, 2016.

[2] X. Liu, C. R. Qi, and L. J. Guibas. FlowNet3D: Learning Scene Flow in 3D Point Clouds. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[3] E. Erçelik, E. Yurtsever, M. Liu, Z. Yang, H. Zhang, P. Topçam, M. Listl, Y. K. Çaylı, and A. Knoll. 3D Object Detection with a Self-supervised Lidar Scene Flow Backbone. In S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, editors, *Computer Vision – ECCV 2022*, pages 247–265, Cham, 2022. Springer Nature Switzerland.

[4] P. Jund, C. Sweeney, N. Abdo, Z. Chen, and J. Shlens. Scalable Scene Flow From Point Clouds in the Real World. *IEEE Robotics and Automation Letters*, 12 2021.

[5] Q. Zhang, Y. Yang, P. Li, O. Andersson, and P. Jensfelt. Seflow: A self-supervised scene flow method in autonomous driving. *arXiv preprint arXiv:2407.01702*, 2024.

[6] K. Vedder, N. Peri, N. Chodosh, I. Khatri, E. Eaton, D. Jayaraman, Y. Liu, D. Ramanan, and J. Hays. ZeroFlow: Scalable Scene Flow via Distillation. In *Twelfth International Conference on Learning Representations (ICLR)*, 2024.

[7] I. Khatri, K. Vedder, N. Peri, D. Ramanan, and J. Hays. I Can't Believe It's Not Scene Flow! In *European Conference on Computer Vision (ECCV)*, 2024.

[8] N. Peri, A. Dave, D. Ramanan, and S. Kong. Towards Long Tailed 3D Detection. *CoRL*, 2022.

[9] T. Weng, S. M. Bajracharya, Y. Wang, K. Agrawal, and D. Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In *Conference on Robot Learning*, pages 192–202. PMLR, 2022.

[10] G. Zhai, X. Kong, J. Cui, Y. Liu, and Z. Yang. FlowMOT: 3D Multi-Object Tracking by Scene Flow Association. *ArXiv*, abs/2012.07541, 2020.

[11] M. Najibi, J. Ji, Y. Zhou, C. R. Qi, X. Yan, S. Ettinger, and D. Anguelov. Motion Inspired Unsupervised Perception and Prediction in Autonomous Driving. European Conference on Computer Vision (ECCV), 2022.

[12] D. Liu, D. Liu, X. Li, S. Lin, H. xie, B. Wang, X. Chang, and L. Chu. Self-supervised multi-frame neural scene flow, 2024. URL https://arxiv.org/abs/2403.16116.

[13] X. Li, J. K. Pontes, and S. Lucey. Neural Scene Flow Prior. *Advances in Neural Information Processing Systems*, 34, 2021.

[14] N. Chodosh, D. Ramanan, and S. Lucey. Re-Evaluating LiDAR Scene Flow for Autonomous Driving. *arXiv preprint*, 2023.

[15] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu. Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection. *arXiv preprint arXiv:1908.09492*, 2019.

[16] K. Vedder and E. Eaton. Sparse PointPillars: Maintaining and Exploiting Input Sparsity to Improve Runtime on Embedded Systems. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[17] N. Peri, J. Luiten, M. Li, A. Osep, L. Leal-Taixe, and D. Ramanan. Forecasting from LiDAR via Future Object Detection. *arXiv:2203.16297*, 2022.

[18] N. Peri, M. Li, B. Wilson, Y.-X. Wang, J. Hays, and D. Ramanan. An empirical analysis of range for 3d object detection. *arXiv preprint arXiv:2308.04054*, 2023.

[19] C. Nalty, N. Peri, J. Gleason, C. Castillo, S. Hu, T. Bourlai, and R. Chellappa. A Brief Survey on Person Recognition at a Distance. 12 2022. doi:10.48550/arXiv.2212.08969.

[20] J. Kim, J. Woo, U. Shin, J. Oh, and S. Im. Flow4D: Leveraging 4D Voxel Network for LiDAR Scene Flow Estimation, 2024. URL https://arxiv.org/abs/2407.07995.

[21] A. Behl, D. Paschalidou, S. Donné, and A. Geiger. Pointflownet: Learning representations for rigid motion estimation from point clouds. In *Int. Conf. Comput. Vis.*, pages 7962–7971, 2019.

[22] I. Tishchenko, S. Lombardi, M. R. Oswald, and M. Pollefeys. Self-supervised learning of non-rigid residual flow and ego-motion. In *Int. Conf. 3D Vis.*, pages 150–159. IEEE, 2020.

[23] Y. Kittenplon, Y. C. Eldar, and D. Raviv. Flowstep3d: Model unrolling for self-supervised scene flow estimation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4114–4123, 2021.

[24] W. Wu, Z. Y. Wang, Z. Li, W. Liu, and L. Fuxin. Pointpwc-net: Cost volume on point clouds for (self-) supervised scene flow estimation. In *Eur. Conf. Comput. Vis.*, pages 88–107. Springer, 2020.

[25] G. Puy, A. Boulch, and R. Marlet. Flot: Scene flow on point clouds guided by optimal transport. In *Eur. Conf. Comput. Vis.*, pages 527–544. Springer, 2020.

[26] R. Li, G. Lin, T. He, F. Liu, and C. Shen. HCRF-Flow: Scene flow from point clouds with continuous high-order CRFs and position-aware flow embedding. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 364–373, 2021.

[27] X. Gu, Y. Wang, C. Wu, Y. J. Lee, and P. Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3254–3263, 2019.

[28] R. Battrawy, R. Schuster, M.-A. N. Mahani, and D. Stricker. RMS-FlowNet: Efficient and Robust Multi-Scale Scene Flow Estimation for Large-Scale Point Clouds. In *Int. Conf. Rob. Aut.*, pages 883–889. IEEE, 2022.

[29] J. Wang, X. Li, A. Sullivan, L. Abbott, and S. Chen. PointMotionNet: Point-Wise Motion Learning for Large-Scale LiDAR Point Clouds Sequences. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4418–4427, 2022.

[30] Q. Zhang, Y. Yang, H. Fang, R. Geng, and P. Jensfelt. DeFlow: Decoder of Scene Flow Network in Autonomous Driving. *ICRA*, 2024.

[31] Y. Lin and H. Caesar. ICP-Flow: LiDAR Scene Flow Estimation with ICP. 2024.

[32] C. Wang, X. Li, J. K. Pontes, and S. Lucey. Neural Prior for Trajectory Estimation. In *CVPR*, pages 6522–6532, 2022. doi:10.1109/CVPR52688.2022.00642.

[33] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla. Nerfies: Deformable Neural Radiance Fields. *ICCV*, 2021.

[34] R. A. Newcombe, D. Fox, and S. M. Seitz. DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, 2015. doi:10.1109/CVPR.2015.7298631.

[35] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, and J. Hays. Argoverse 2: Next Generation Datasets for Self-driving Perception and Forecasting. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*, 2021.

[36] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[37] X. Li, J. Zheng, F. Ferroni, J. K. Pontes, and S. Lucey. Fast Neural Scene Flow. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9878–9890, October 2023.

[38] K. Vidanapathirana, S.-F. Chng, X. Li, and S. Lucey. Multi-body neural scene flow. In *2024 International Conference on 3D Vision (3DV)*. IEEE, 2024.

[39] 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. *Commun. ACM*, 65(1):99–106, dec 2021. ISSN 0001-0782.

[40] N. Chodosh, A. Madan, D. Ramanan, and S. Lucey. Simultaneous Map and Object Reconstruction, 2024. URL https://arxiv.org/abs/2406.13896.

[41] T. Khurana, P. Hu, D. Held, and D. Ramanan. Point Cloud Forecasting as a Proxy for 4D Occupancy Forecasting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[42] B. Agro, Q. Sykora, S. Casas, T. Gilles, and R. Urtasun. UnO: Unsupervised Occupancy Fields for Perception and Forecasting. In *CVPR*, 2024.

[43] S. Venkatesh, B. Bianchini, A. Aydinoglu, and M. Posa. Sampling-Based Model Predictive Control for Contact-Rich Manipulation. In *IROS 2023 Workshop on Leveraging Models for Contact-Rich Manipulation*, 2023.

[44] S. Ramasinghe, H. Saratchandran, V. Shevchenko, A. Long, and S. Lucey. On the Optimality of Activations in Implicit Neural Representations, 2024. URL https://openreview.net/forum?id=0Lqyut1y7M.

[45] S.-F. Chng, S. Ramasinghe, J. Sherrah, and S. Lucey. Gaussian Activated Neural Radiance Fields for High Fidelity Reconstruction and Pose Estimation. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIII*, page 264–280, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-19826-7.

[46] W. Hu, X. Gao, X. Li, S. Zhao, X. Cun, Y. Zhang, L. Quan, and Y. Shan. DepthCrafter: Generating Consistent Long Depth Sequences for Open-world Videos. *arXiv preprint arXiv:2409.02095*, 2024.

[47] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NeurIPS'18, page 6572–6583, Red Hook, NY, USA, 2018.

[48] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu. Liquid Time-constant Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:7657–7666, May 2021.

[49] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[50] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European Conf. on Computer Vision (ECCV)*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, Oct. 2012.

A Additional Figures

(a) Bird trajectory via Euler integration from takeoff (b) Bird being tracked

Figure 8: EulerFlow is able to track the bird over 20 frames by performing Euler integration starting from takeoff until it loses all point cloud lidar returns.

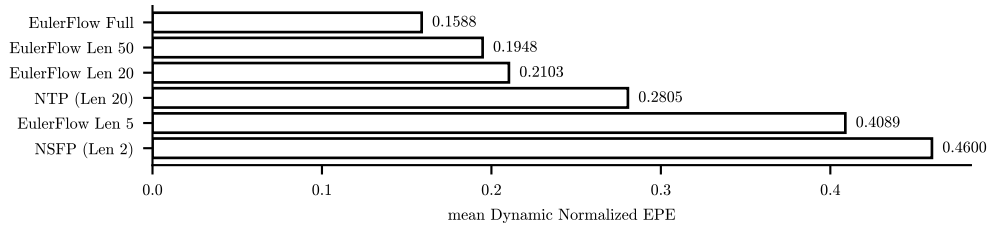


Figure 9: Mean Dynamic Normalized EPE of EulerFlow for various sequence lengths on the Argoverse 2 val split, compared against representative baselines. These results demonstrate that EulerFlow improves with sequence length; however, at a sequence length of 20, our method significantly outperforms NTP, suggesting that EulerFlow's performance cannot solely be attributed to longer sequence modeling.

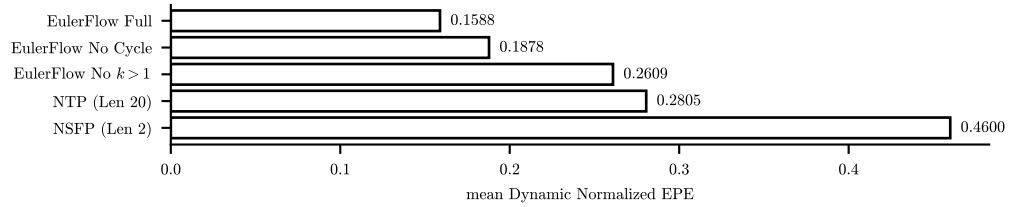


Figure 10: Mean Dynamic Normalized EPE of EulerFlow for various losses on the Argoverse 2 val split, compared against representative baselines. These results demonstrate that EulerFlow's multi-observation optimization objectives significantly improve overall performance.

B Additional results

B.1 How does the choice of learnable function class and design of encodings impact EulerFlow?

EulerFlow at its core is an optimization loop over a simple, feedforward ReLU-based multi-layer perception inherited from Neural Scene Flow Prior [13]. How does this choice of learnable function class impact the performance of EulerFlow? To better understand these design choices we examine the choice of non-linearity and time feature encoding.

One of Li et al.'s core theoretical contributions demonstrates that NSFP's ReLU MLP is a good prior for scene flow because it represents a smooth learnable function class, and scene flow is often locally smooth with respect to input position. However, unlike NSFP, EulerFlow is fitting flow over a full ODE; while it seems reasonable to assume that this ODE is typically also locally smooth, cases like adjacent cars moving rapidly in opposite directions may benefit from the ability to model higher frequency, less locally smooth functions. To test this hypothesis, we ablate EulerFlow by replacing its normalized time with higher frequency sinusoidal time embeddings (mirroring Wang

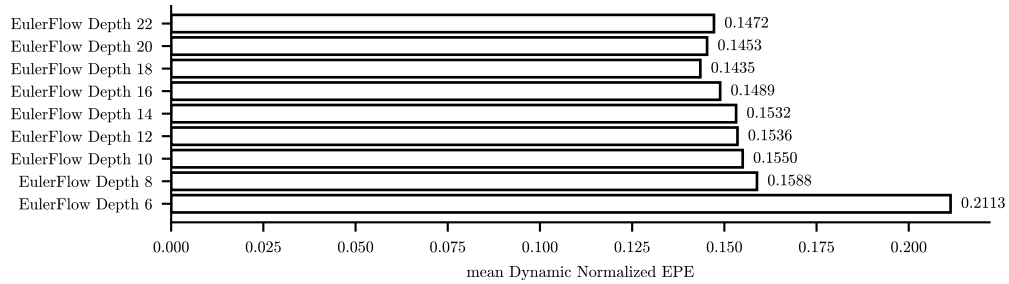
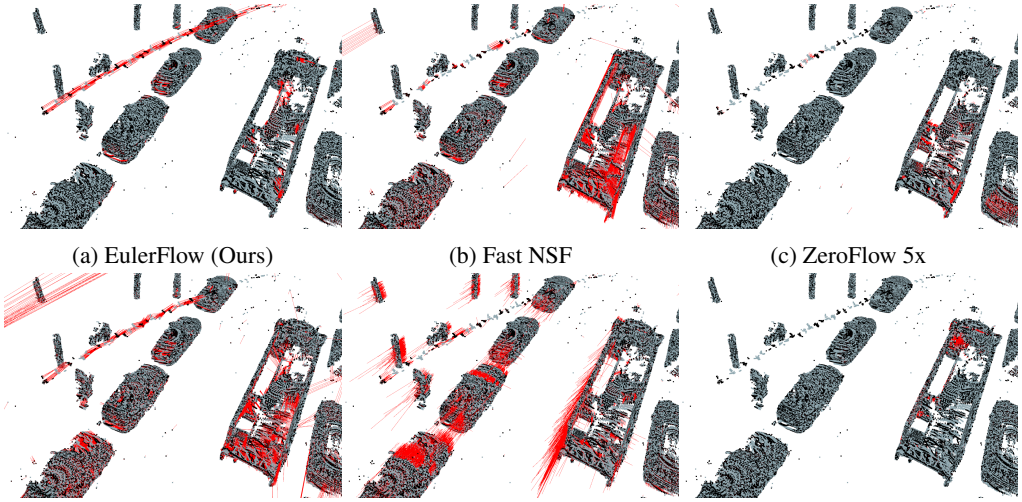


Figure 11: Mean Dynamic Normalized EPE of EulerFlow on the Argoverse 2 val split for different neural prior capacities. Shallow networks underfit the PDE, while deeper networks overfit to noise in the optimization objectives.



(a) EulerFlow (Ours) (b) Fast NSF (c) ZeroFlow 5x

(d) Liu et al. (e) NSFP (f) Ground Truth

Figure 12: Visualization of EulerFlow compared to prior art for the same scene as Figure 1a and Figure 8a. EulerFlow is able to extract the bird's trajectory; however, all other methods except Liu et al. fail to recognize this motion, and Liu et al.'s flow is marred by severe scene artifacts. The bird is outside the labeled object taxonomy, and so its motion is unlabeled in the ground truth (Figure 12f).

et al.'s proposed time embedding for NTP), as well as try other popular non-linearities like SinC [44] and Gaussian [45] from the coordinate network literature. Figure 14 features negative results on these ablations across the board; Gaussians were unable to converge due the extremely high frequency representation triggering early stopping, while the use of SinC and higher frequency time embeddings both resulted in worse overall performance, indicating that Li et al.'s smooth function prior does indeed seem appropriate for EulerFlow's neural prior.

B.2 EulerFlow with Monocular Depth Estimates

While EulerFlow only consumes point clouds, we can leverage RGB-based video monocular depth estimators to fit scene flow. In Figure 15, we use DepthCrafter [46] to generate a point cloud from the raw RGB of the tabletop video from Figure 13, Row 4.

B.3 How Does EulerFlow Fail?

As we discuss in Section 6.1, EulerFlow does not understand projective geometry — its optimization losses use Chamfer Distance which directly associates points, sometimes resulting in moving shadows on background objects. To demonstrate this, we select a particularly egregious example in Figure 16, featuring a frame from the jack being thrown across the table. Due to the moving shadow cast

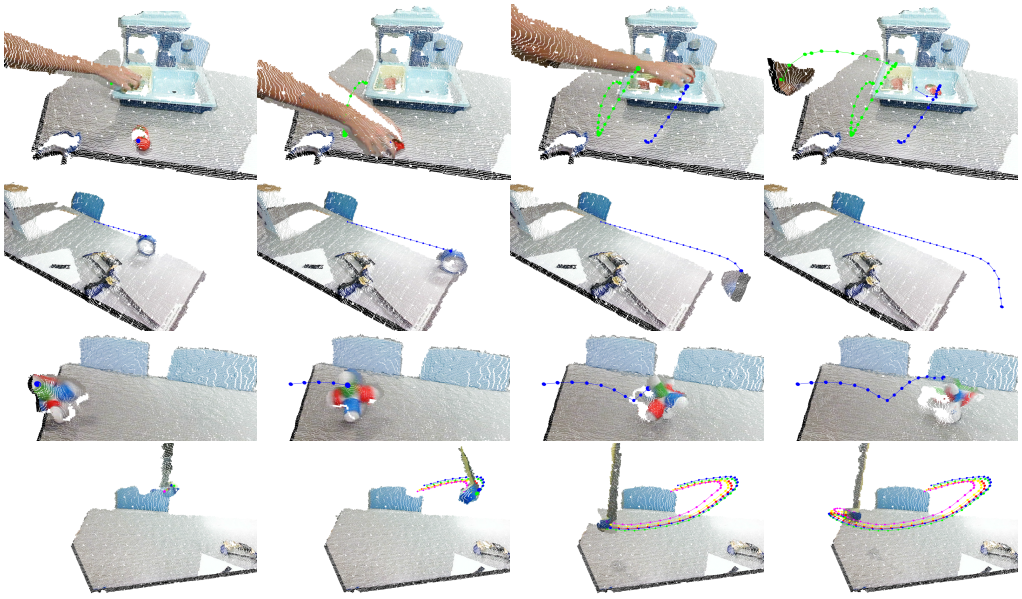


Figure 13: Visualizations of EulerFlow's emergent 3D point tracking behavior that demonstrate the quality of its PDE estimate. Row 1 depicts tracking a tomato placed in the sink by a human hand; note the point does not move despite the hand grasping the tomato. Row 2 depicts tracking of painters tape rolling off a table; EulerFlow is able to estimate its trajectory even after it disappears out of frame. Row 3 depicts tracking of the motion of a jack commonly used in tabletop manipulation experiments [43]. Row 4 depicts tracking of a tennis ball taped to a flexible rod. All tracks are produced by Euler integration through the estimated PDE from the initial conditions shown in the left column. Note that point clouds are shown in color for visualization purposes only; RGB is not used during optimization.

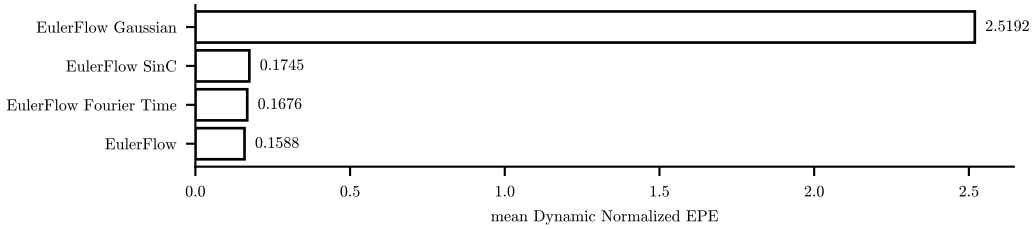


Figure 14: Mean Dynamic Normalized EPE of EulerFlow on the Argoverse 2 val split for less-smooth configurations of its learnable function class. These results indicate that the smoothness of the ReLU non-linearity proposed by Li et al. transfers well to EulerFlow.

by the jack onto the table, EulerFlow incorrectly assigns flow to the table surface nearby the jack, particularly on the leading edge, even though the table surface is stationary.

C FAQ

C.1 What datasets did you pretrain on?

EulerFlow is not pretrained on any datasets. It is a test-time optimization method (akin to NeRFs), and as we show with our tabletop data, this means it runs out-of-the-box on arbitrary point cloud data.

C.2 Why didn't you use a Neural ODE or a Liquid Neural Network?

Neural ODEs [47] take variable size and number of steps in latent space to do inference; imagine a ResNet that can use an ODE solver to dynamically scale the impact of the residual block, as well as decide the number of residual blocks. They are not a function class specially designed to fit derivative

Figure 15: Visualizations of EulerFlow's emergent 3D point tracking behavior on monocular depth estimates from DepthCrafter [46]. Interactive visualizations available at `vedder.io/eulerflow`.

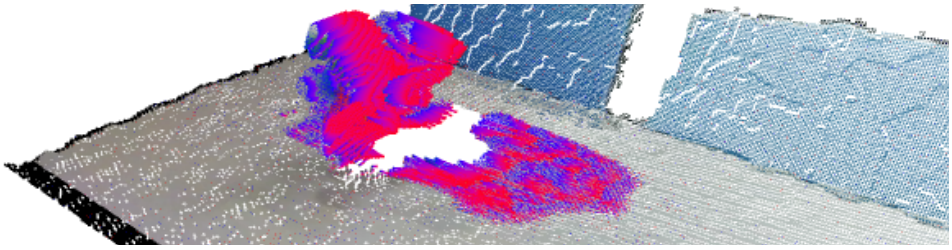


Figure 16: Visualizations of one of the failure modes of EulerFlow where flow is predicted on the edges of the moving "shadow" in the point cloud. Interactive visualizations available at `vedder.io/eulerflow`.

estimates well. Similar to Neural ODEs, Liquid Neural Networks [48] focus on the same class of problems and are similarly not applicable.

C.3 Why didn't you do experiments on FlyingThings3D / <simulated dataset>?

Most popular synthetic datasets do not contain long observation sequences [49, 50], but instead include standalone frame pairs. Our method leverages the long sequence of observations to refine our neural estimate of the true ODE. Indeed, on two frames, EulerFlow collapses to NSFP.

More importantly, these datasets are also not representative of real world environments. To quote Chodosh et al.: "[FlyingThings3D has] unrealistic rates of dynamic motion, unrealistic correspondences, and unrealistic sampling patterns. As a result, progress on these benchmarks is misleading and may cause researchers to focus on the wrong problems." Khatri et al. also make this point by highlighting the importance of meaningfully breaking down the object distribution during evaluation identify performance on rare safety-critical categories. FlyingThings3D does not have meaningful semantics; it's not obvious what things even matter or how to appropriately break down the scene.

Instead, we want to turn our attention to the sort of workloads that *do* clearly matter — describing motion in domains like manipulation or autonomous vehicles, where it seems clear that scene flow, if solved, will serve as powerful primitive for downstream systems. This is why we performed qualitative experiments on the tabletop data we collected ourselves; to our knowledge, no real-world dynamic datasets of this nature exist with ground truth annotations, but we want to emphasize that EulerFlow works in such domains, and consequently EulerFlow and other Scene Flow via ODE-based methods can be used as a primitive in these real world domains.

D EulerFlow implementation details

Our neural prior θ is a straightforward extension to NSFP's coordinate network prior[6]; however, instead of taking a 3D space vector (positions $X, Y, Z \in \mathbb{R}$) as input, we encode a 5D space-time-direction vector: positions $X, Y, Z, \in \mathbb{R}$, sequence normalized time $t \in [-1, 1]$ (i.e. the point cloud time scaled to this range), and direction $d \in \{\texttt{BWD} = -1, \texttt{FWD} = 1\}$. This simple encoding scheme enables description of arbitrary regions of the ODE, allowing for the ODE to be queried at frequencies

[6]Hyperparameters (e.g. filter width of 128) of NSFP's prior are kept fixed, except for depth (Section 5.2.3).

different from the sensor frame rate. Euler integration enables simple implementation of multi-step forward, backward, and cyclic consistency losses without extra bells and whistles. For efficiency, we use Euler integration with Δt set as the time between observations for our ODE solver, enabling support for arbitrary sensor frame rates, and set the cycle consistency balancing term $\alpha = 0.01$ and optimization window $W = 3$ for all experiments.

E EulerFlow's ODE Derivation

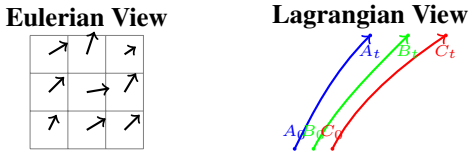


Figure 17: Comparison of Eulerian and Lagrangian descriptions of 2D flow. An Eulerian view characterizes a flow field via instantaneous velocities at many different points, while a Lagrangian view characterizes a flow field via trajectories of many different particles across time. Both approaches are valid ways of describing an underlying flow field, and with sufficient characterization one view can be readily converted to another, but the Lagrangian view relies on a definition of the definition of consistent canonical frame.

E.1 Formulating the ODE

Given a (possibly moving) particle in some canonical frame (i.e. time 0), we define a function $L(x_0, y_0, z_0, t)$ that can describe its location at an arbitrary future time t, i.e. a Lagrangian description of motion (Figure 17).

$$L(x_0, y_0, z_0, t) = x_t, y_t, z_t \tag{4}$$

For notational clarity to access x_t, y_t, z_t individually, we can define

$$L_x(x_0, y_0, z_0, t) = x_t \tag{5}$$
$$L_y(x_0, y_0, z_0, t) = y_t \tag{6}$$
$$L_z(x_0, y_0, z_0, t) = z_t \tag{7}$$

Similarly, we can define $F(x_t, y_t, z_t, t)$ to describe the instantaneous velocity of a point x_t, y_t, z_t at some arbitrary time t, i.e. a Eulerian description of motion (Figure 17).

$$\frac{dL(x_0, y_0, z_0, t)}{dt} = \frac{dL}{dt} = \left(\frac{dL_x}{dt}, \frac{dL_y}{dt}, \frac{dL_z}{dt}\right) = F(x_t, y_t, z_t, t) \tag{8}$$

F is defined in terms of the total derivative of L with respect to t, as x_0, y_0, z_0 are initial conditions that do not vary with time (i.e. $\frac{dL}{dt} = \frac{\partial L}{\partial t} + \frac{\partial L}{\partial x_0}\frac{dx_0}{dt} + \frac{\partial L}{\partial y_0}\frac{dy_0}{dt} + \frac{\partial L}{\partial z_0}\frac{dz_0}{dt} = \frac{\partial L}{\partial t}$, as $\frac{dx_0}{dt} = \frac{dy_0}{dt} = \frac{dz_0}{dt} = 0$). We can exactly define L recursively in terms of the initial conditions and F, i.e.

$$L(x_0, y_0, z_0, t) = (x_0, y_0, z_0) + \int_0^t F(L_x(x_0, y_0, z_0, \tau), L_y(x_0, y_0, z_0, \tau), L_z(x_0, y_0, z_0, \tau), \tau) d\tau \tag{9}$$

or, more compactly,

$$L(x_0, y_0, z_0, t) = (x_0, y_0, z_0) + \int_0^t F(x_\tau, y_\tau, z_\tau, \tau) d\tau \tag{10}$$

Our function L can thus be defined as a multi-dimensional ODE in terms of F with initial conditions x_0, y_0, z_0.

E.2 Arbitrary start and end times from the Eulerian formulation

In the above derivation, L requires that a moving point be defined in terms of a canonical frame defined at time 0, as is common in the deformation in reconstruction literature. However, the Eulerian formulation has no such requirement, allowing us to select arbitrary start and end times across different point queries. To showcase this, we can query F to extract the trajectory of a particle at t across the range $[t, t']$ starting at x_t, y_t, z_t simply by changing the range of the integral in Equation 10, i.e.

$$E(x_t, y_t, z_t, t, t') = (x_t, y_t, z_t) + \int_t^{t'} F(x_\tau, y_\tau, z_\tau, \tau) d\tau \tag{11}$$

While E and L appear similar on their face, E is strictly more flexible than L. In principle you could choose to redefine L to use t as the time for your canonical frame, but this is a *global* choice; you cannot do this on a per-query basis. However, with E's Eulerian framing, we can extract a different point's trajectory from the entirely different range $t^\dagger$ to $t^\ddagger$ (i.e. $E(x_{t^\dagger}, y_{t^\dagger}, z_{t^\dagger}, t^\dagger, t^\ddagger)$) without concern for a canonical frame definition. It need not even be the case that $t < t'$; indeed, this extraction works even if $t > t'$, i.e. extracting the backwards trajectory through time.

E.3 Euler Integration to approximately solve the ODE

If F is of arbitrary form and we want to compute the concrete values of L, we cannot exactly compute the continuous integral from 0 to t; we must approximate this with finite differences. Thus, we split the time range 0 to t into k steps, where each step is of size $\frac{t}{k}$. Thus, we can again define L via recursion, but this time explicitly.

$$L(x_0, y_0, z_0, 0) = (x_0, y_0, z_0) \tag{12}$$

$$L(x_0, y_0, z_0, \tau + \frac{t}{k}) \approx L(x_0, y_0, z_0, \tau) + \frac{t}{k} \cdot F(x_\tau, y_\tau, z_\tau, \tau), \tag{13}$$

or directly without recursion,

$$L(x_0, y_0, z_0, t) \approx (x_0, y_0, z_0) + \sum_{n=1}^{k} \frac{t}{k} \cdot F(x_{n\frac{t}{k}}, y_{n\frac{t}{k}}, z_{n\frac{t}{k}}, n\frac{t}{k}) \tag{14}$$

This finite difference solving approach is Euler integration.

E.4 Estimating the flow field with EulerFlow's neural prior

For a given scene, we do not have access to L or F directly; these are are the *true* functions that uniquely characterize the underlying motion of the scene that we are trying to estimate. For EulerFlow, we represent our estimate of the scene's flow field F with a neural prior, θ, i.e.

$$F(x, y, z, t) \approx \theta(x, y, z, t) \tag{15}$$

and thus

$$L(x_0, y_0, z_0, t) \approx (x_0, y_0, z_0) + \sum_{n=1}^{k} \frac{t}{k} \cdot \theta(x_{n\frac{t}{k}}, y_{n\frac{t}{k}}, z_{n\frac{t}{k}}, n\frac{t}{k}) \tag{16}$$

and, using the arbitrary start and end definition from Appendix E.2, with k steps from the range t to t' and $\delta = \frac{t'-t}{k}$

$$E(x_t, y_t, z_t, t, t') \approx E_\theta(x_t, y_t, z_t, t, t') = (x_t, y_t, z_t) + \sum_{n=1}^{k} \delta \cdot \theta(x_{n\delta+t}, y_{n\delta+t}, z_{n\delta+t}, n\delta + t) \quad (17)$$

This formulation makes EulerFlow highly flexible, enabling optimization of θ's estimate of F with objectives that take either an Eulerian view (directly on θ via Equation 15) or a Lagrangian view (on point rollouts for arbitrary start and end ranges via Equation 17).