# Supplementary Material for Fast Neural Scene Flow

Xueqian Li[*1] Jianqiao Zheng[1] Francesco Ferroni[*2] Jhony Kaesemodel Pontes[*3] Simon Lucey[*1]
[1]The University of Adelaide [2]NVIDIA [3]Latitude AI

## 1. Experiments

### 1.1. Speedup in network architectures

**Positional encodings.** Positional encodings (PEs) are usually used in coordinate networks [7] to increase the bandwidth of the input by multiple (random) frequencies. With PEs, coordinate network converges much faster and achieves better performance [12]. Moreover, a recent paper [15] points out that the success of PE attributes to the rank increase of the input: the deepness of the neural network is to increase the rank of the embedding and align the embedding space to the output space. However, if the rank of the input is higher, the network can be shallower. Therefore, [15] uses a more complex PE to dramatically increase the rank of the input, thus the followed deep non-linear network can be replaced by a shallow linear function.

**Complex PE-based linear model.** In our paper, we implemented a complex positional encoding (PE) [15]-based linear model to test the efficiency of simplifying network architectures. While simple PE refers to a simple concatenation of the encoding in each input dimension, complex PE [15] is a more complicated encoding that computes the Kronecker product of the per-axis encoding. As mentioned in [15], one reason behind the success of the deep network is that it increases the rank of the low-rank input. Therefore, if the input to the network has a high rank, the network can be shallower accordingly. To increase the rank of the input, we use a complex encoding instead of a deep network. The rank of the complex encoding is

$$\begin{aligned}&\operatorname{Rank}\left(\phi(\mathbf{p}_x)\otimes\phi(\mathbf{p}_y)\otimes\phi(\mathbf{p}_z)\right)=\\&\quad\operatorname{Rank}\left(\phi(\mathbf{p}_x)\right)\operatorname{Rank}\left(\phi(\mathbf{p}_y)\right)\operatorname{Rank}\left(\phi(\mathbf{p}_z)\right),\end{aligned}\tag{1}$$

which achieves full rank that allows us to only use a linear layer $\mathbf{W}$ as a follow-up embedder.

The advantage of a complex PE lies in two aspects: first, with a linear layer, the problem can be solved analytically in many cases; second, if the closed-form solution is difficult to obtain, using a linear layer in iterative solvers, such as gradient descent-based methods, will converge faster. Similar to frequency-based encodings [12], shift-based encodings like Gaussian or Triangle wave also work similarly well [15]. These shift-based encodings involve very few parameters for each sample point due to sparsity, while a deep network requires significant amounts of parameters.

To reconstruct the signal $\mathbf{S}$, we optimize

$$\arg\min_{\mathbf{W}}\left\|\operatorname{vec}(\mathbf{S})-\left(\phi(\mathbf{p}_x)\otimes\phi(\mathbf{p}_y)\otimes\phi(\mathbf{p}_z)\right)^T\operatorname{vec}(\mathbf{W})\right\|_2^2.\tag{2}$$

When the coordinate is separable along each axis (*e.g.*, 2D image), using Kronecker product, we have a closed-form solution as

$$\mathbf{W}=\phi(\mathbf{p}_x)^{-1}\mathbf{S}\phi(\mathbf{p}_y)^{-T}\phi(\mathbf{p}_z)^{-T}.\tag{3}$$

With all the complex PE theory in hand, it is nontrivial to implement it in a scene flow problem, and to the best of our knowledge, we are the first to apply complex PE to real-word large-scale data. To employ complex PE in the scene flow problem, we first replace the non-linear multi-layer perceptrons (MLPs) in NSFP with a linear layer parameterized by $\mathbf{W}\in\mathbb{R}^{W_xW_yW_z\times 3}$—$W_x$, $W_y$, and $W_z$ are encodings in each dimension and 3 is the dimension of the flow—and encode the input coordinates in complex PE form. The flow represented by MLPs can then be modified as

$$\mathbf{f}=g\left(\mathbf{p};\,\mathbf{W}\right)=\left(\phi(\mathbf{p}_z)\otimes\phi(\mathbf{p}_y)\otimes\phi(\mathbf{p}_x)\right)\operatorname{vec}\left(\mathbf{W}\right)\tag{4}$$

where $\phi\left(\cdot\right)$ is the encoder, $\mathbf{p}_x$, $\mathbf{p}_y$, $\mathbf{p}_z$ are the sample points in $x$, $y$, $z$ coordinates.

**Blending function.** However, in the 3D point cloud case, the unordered points are not separable in each axis. A blending function B is introduced to interpolate the points and avoid the computation of large naive complex PE. Please note that the unordered point cloud has non-separable coordinates. According to [15], the non-separable-coordinate problem can be approximated by a blending function and an encoding of virtual separable grid points. The blending approximation is $\phi(\mathbf{p}_z)\otimes\phi(\mathbf{p}_y)\otimes\phi(\mathbf{p}_x)\approx\mathrm{B}(\mathbf{p};\phi)\phi(\mathbf{z})\otimes\phi(\mathbf{y})\otimes\phi(\mathbf{x})$,

*Part of the work was done while at Argo AI. Corresponding e-mail: xueqian.li@adelaide.edu.au.

where $\mathbf{x} \in \mathbb{R}^{W_x}$, $\mathbf{y} \in \mathbb{R}^{W_y}$ and $\mathbf{z} \in \mathbb{R}^{W_z}$ are virtual grid points. Using such approximation, the computation of complex PE of grid points is as easy and fast as a matrix multiplication due to the property of the Kronecker product. Intuitively, the blending matrix B can be viewed as a matrix consisting of non-linear interpolation coefficients that depend on the encoding function $\phi(\cdot)$. It is large but sparse, *i.e.*, there are only 8 non-zero values on each row (corresponding to the 8 neighboring grid points of the query point), we can index the matrix efficiently by only querying the non-zero entries. Meanwhile, different from [15], grids here have physical meanings and their size can be adjusted.

Therefore, the scene flow becomes

$$\mathbf{f} \approx \mathrm{B}(\mathbf{p}; \phi)\mathrm{vec}\left(\phi(\mathbf{x})\mathbf{W}\phi(\mathbf{z})^T\phi(\mathbf{y})^T\right), \tag{5}$$

where $\phi(\mathbf{x})\mathbf{W}\phi(\mathbf{z})^T\phi(\mathbf{y})^T$ is a simple notation for $n$-mode multiplication. And the scene flow optimization is

$$\arg\min_{\mathbf{W}} \left\| \mathbf{f} - \mathrm{B}\left(\phi(\mathbf{p}_x) \otimes \phi(\mathbf{p}_y) \otimes \phi(\mathbf{p}_z)\right)^T \mathrm{vec}(\mathbf{W}) \right\|_2^2. \tag{6}$$

We therefore solve $\mathbf{W}$ using gradient descent and distance transform loss as

$$\begin{aligned} \mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{\mathbf{p} \in \mathcal{S}_1} \mathrm{D}\left(\mathbf{p} + \mathrm{B}\left(\mathbf{p}; \phi\right)\right. \\ \left. \mathrm{vec}\left(\phi(\mathbf{x})\mathbf{W}\phi(\mathbf{z})^T\phi(\mathbf{y})^T\right), \mathcal{S}_2\right). \end{aligned} \tag{7}$$

Compared to a $L$ layer network of width $W$ used in NSFP to process $N$ sample points, a linear layer of size $W_x \times W_y \times W_z$ from the Kronecker product speeds up the network significantly from $\mathcal{O}(NW^2L)$ to $\mathcal{O}(8N + 3W_xW_yW_z\left(W_x + W_y + W_z\right))$. We further constrain the optimization by applying an explicit total variation (TV) regularizer on $\mathbf{W}$ as:

$$\begin{aligned} \mathrm{TV}\left(\mathbf{W}\right) = \frac{1}{(W_x - 1)(W_y - 1)(W_z - 1)} \\ \sum_{i=0}^{W_x-2} \sum_{j=0}^{W_y-2} \sum_{k=0}^{W_z-2} \sqrt{(d\mathbf{x}_{i,j,k})^2 + (d\mathbf{y}_{i,j,k})^2 + (d\mathbf{z}_{i,j,k})^2}, \end{aligned} \tag{8}$$

where $\mathrm{dx}_{i,j,k} = \mathbf{W}_{i,j,k} - \mathbf{W}_{i+1,j,k}$, $\mathrm{dy}_{i,j,k} = \mathbf{W}_{i,j,k} - \mathbf{W}_{i,j+1,k}$, $\mathrm{dz}_{i,j,k} = \mathbf{W}_{i,j,k} - \mathbf{W}_{i,j,k+1}$. In all, the loss function of complex PE model is (with TV)

$$\begin{aligned} \mathcal{L}\left(\mathbf{W}\right) = \sum_{\mathbf{p} \in \mathcal{S}_1} \mathrm{D}\left(\mathbf{p} + \mathrm{B}\left(\mathbf{p}\right) \mathrm{vec}\left(\phi(\mathbf{x})\mathbf{W}\phi(\mathbf{z})^T\right.\right. \\ \left.\left. \phi(\mathbf{y})^T\right), \mathcal{S}_2\right) + \frac{\lambda}{2}\mathrm{TV}\left(\mathbf{W}\right). \end{aligned} \tag{9}$$

## 1.2. Speedup in point correspondence search

Before the deployment of the distance transform (DT), we explored other strategies to speed up the point correspondence search in Chamfer distance.

**Build a k-d tree to search nearest neighboring points.** The point distance function D is computationally intensive, as a set of point-to-point correspondences needs to be optimized in each optimization step. One speedup is to construct a k-d tree to accelerate the nearest neighbor search which reduces the computation complexity of point correspondence search from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. Since the target point cloud $\mathcal{S}_2$ is fixed, we only need to pre-build the k-d tree for $\mathcal{S}_2$ once. However, the source point cloud $\mathcal{S}_1$ is deformed in each optimization step, making the pre-build of the source point cloud k-d tree happens in every iteration, not to mention that the per-iteration k-d tree query is another computation overhead when the number of points is big.

**Randomly sample points.** Stochastic gradient descent (SGD) [10] is now broadly used in large-scale learning problems. It approximates the actual gradient descent by only computing the gradient of a randomly selected subset of the original dataset at each iteration. However, it can achieve relatively faster updates and guarantee a satisfied global convergence, especially when dealing with large-scale high-dimensional optimization [1].

Inspired by the idea of SGD, we choose to randomly subsample points of the dense point cloud at each iteration. Analogous to SGD, each individual point is viewed as sample data. A naive sampling strategy is to sample a fixed number of points. Instead, we develop sampling strategies based on the number of iterations or the decreasing percentage of the loss function. We sample fewer points at the beginning of the optimization when the point correspondences are noisy, and gradually increase the number of sampled points when the optimization becomes better constrained and finds better correspondences.

However, we also noticed that point sampling is not a practical strategy when applied to real-world problems, such as autonomous driving scenarios, where all points are needed to get sufficient information for detailed non-rigid motions.

**Reduce the frequency of updating correspondence.** Although Eq. (2) of the main paper optimizes network weights (scene flow) through an explicit point distance function, the point correspondence optimization is implicitly included. We have mentioned that the optimization of the point correspondence and the scene flow are highly entangled. We cannot easily get a good scene flow estimation even given the optimal point correspondences.

Instead of separating the scene flow and correspondence optimization, we reduce the updating frequency of the point correspondences from every single iteration to several iterations. To guarantee a good initialization, we initially consecutively update correspondences for a fixed number of iterations.

Table 1. **Additional computation time and performance on Waymo Open Scene Flow dataset.** The upper tabular between **blue bars** are experiments with the full point cloud, and the lower tabular between **orange bars** are experiments with only 8,192 points. Corr. / k-d tree / DT query denotes correspondence search, k-d tree-based correspondence search, or DT query.

| Method | $\mathcal{E}$ $(m)\downarrow$ | $Acc_5$ $(\%)\uparrow$ | $Acc_{10}$ $(\%)\uparrow$ | $\theta_\epsilon$ $(rad)\downarrow$ | $t\ (ms)\downarrow$ | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Pre-compute | Corr. / k-d tree / DT query | Network | Total |
| NSFP (baseline) | 0.118 | 74.16 | 86.70 | 0.300 | — | 43.1 [15036] | 2.38 [904] | 18.39 s |
| Baseline (k-d tree CD) | 0.104 | 74.13 | 86.81 | 0.296 | 4.36 | 15.24 [5283] \| **2.8**× | 2.27 [838] \|**1.05**× | 8.51 s \|**2.16**× |
| Baseline (k-d tree CD, linear) | 0.101 | 70.14 | 86.24 | 0.315 | 10.23 | 12.92 [2349] \| **3.3**× | 1.39[262] \|**1.71**× | 4.15 s \|**4.43**× |
| PointPWC-Net [14] | 4.109 | 0.05 | 0.36 | 1.742 | — | — | — | 185 ms \|**1.32**× |
| FlowStep3D [4] | 0.753 | 0.01 | 0.09 | 1.212 | — | — | — | 725 ms \|**5.18**× |
| PV-RAFT [13] | 10.675 | 0.03 | 0.13 | 1.794 | — | — | — | 505 ms \|**3.61**× |
| R3DSF [3] | 0.414 | 35.47 | 44.96 | 0.527 | — | — | — | 140 ms |
| Ours (8,192 pts) | **0.106** | **77.53** | **88.99** | **0.329** | 35.22 | 0.23 [6.5] \|**496**× | 2.60 [76] \|**1.8**× | **121 ms** \|**1.16**× |

Table 2. **Additional computation time and performance on Argoverse Scene Flow dataset.**

| Method | $\mathcal{E}$ $(m)\downarrow$ | $Acc_5$ $(\%)\uparrow$ | $Acc_{10}$ $(\%)\uparrow$ | $\theta_\epsilon$ $(rad)\downarrow$ | $t\ (ms)\downarrow$ | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Pre-compute | Corr. / k-d tree / DT query | Network | Total |
| NSFP (baseline) | 0.078 | 69.46 | 86.22 | 0.253 | — | 17 [5901] | 2.31 [848] | 8.38 s |
| Baseline (k-d tree CD) | 0.078 | 69.14 | 85.99 | 0.253 | 3.96 | 11.3 [4063] \| **1.5**× | 2.28 [830] \|**1.0**× | 6.25 s \|**1.34**× |
| Baseline (k-d tree CD, linear) | 0.071 | 68.72 | 86.39 | 0.288 | 8.87 | 9.36 [1701] \| **1.8**× | 1.41 [253] \|**1.6**× | 3.09 s \|**2.71**× |
| PointPWC-Net [14] | 5.600 | 0.03 | 0.18 | 1.179 | — | — | — | 186 ms \|**1.65**× |
| FlowStep3D [4] | 0.845 | 0.01 | 0.08 | 1.860 | — | — | — | 729 ms \|**6.45**× |
| PV-RAFT [13] | 10.745 | 0.02 | 0.10 | 1.517 | — | — | — | 504 ms \|**4.46**× |
| R3DSF [3] | 0.417 | 32.52 | 42.52 | 0.551 | — | — | — | **113 ms** |
| Ours (8,192 pts) | 0.118 | **69.93** | **83.55** | 0.352 | 41.57 | 0.22 [6.33] \|**214**× | 2.51 [72.69] \| **1.9**× | 124 ms \|**1.10**× |

However, the correspondence sampling strategy is unfavorable due to a considerable performance compromise.

## 1.3. Implementation details

We provide more implementation details for our method. Further details will be provided upon code release.

**Datasets.** We followed [5, 8] to create the pseudo scene flow labels, and removed ground points according to each dataset. Note that we used the raw point cloud from the lidar sensor and did not crop the data to a small range.

**Truncated Chamfer distance.** We used a truncated Chamfer distance loss for our baseline implementation as mentioned in the original NSFP [5] that is unbiased on extreme points. Practically, we chose $2m$ as a threshold to eliminate large point distance.

**Complex positional encoding.** Since point clouds are non-separable in 3D space, we first encoded the separable 3D virtual voxel vertices using shifted Gaussian encoders as depicted in Fig. 3 of the main paper. Since the 3D space in autonomous driving scenarios is large, we empirically found that a relatively larger voxel size (*e.g.*, $2m$ or $5m$ for autonomous driving scene flow datasets) that constrains the motions as rigid as possible within a larger local region is more suitable to encode scene flow. The choice of the Gaussian sigma also depends on the voxel size. Generally, the sigma of Gaussian encoding should be twice larger than the voxel size. For example, for a voxel size of $2m$, $\sigma > 4$ is favored. Note that the Gaussian sigma and the voxel size can be adjusted within a small range.

## 1.4. Additional results

We provide additional results on Waymo Open and Argoverse scene flow datasets in Tab. 1 and Tab. 2 respectively.

We show how k-d tree-based correspondence search for CD loss speeds up the optimization, yet remains less effective, which indicates the inherent computation cost of correspondence search in CD loss cannot be easily solved using engineering techniques. The performance of the linear model drops by a large margin, suggesting that it is a less favorable choice for scene flow estimation.

## 1.5. Performance gap of learning methods

The performance of learning-based methods such as PointPWC-Net [14], FlowStep3D [4], PV-RAFT [13], FLOT [9] is inferior compared to non-learning-based methods (shown in Tab. 1 and Tab. 2, between the **orange** bar). As discussed in the main paper, the performance

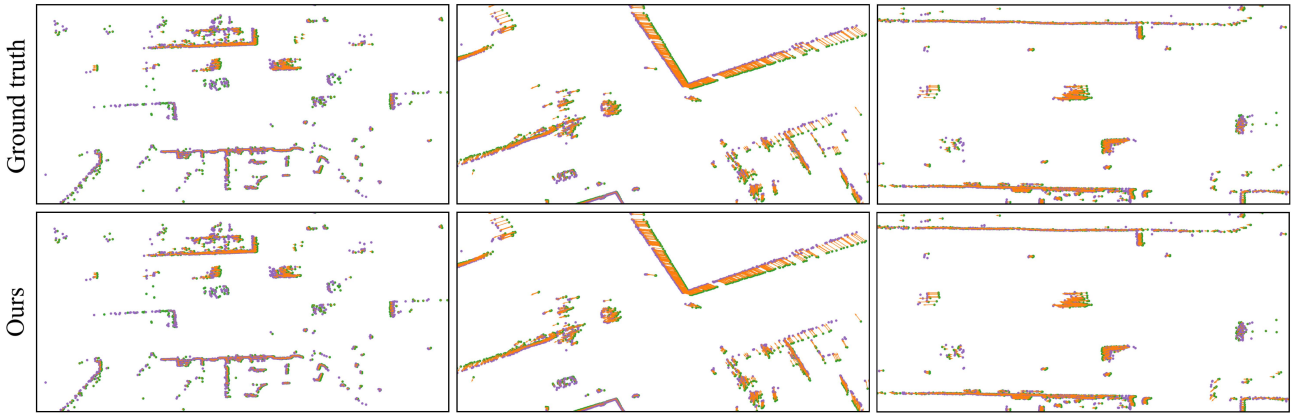



Figure 1. Visual results of the 2D scene flow estimation using our method on the Argoverse scene flow dataset. **Green** points are source, **purple** points are target, and **orange** arrows represent the flow vectors.

Table 3. Performance of distance transform with different grid cell sizes on a 2D BEV scene from Argoverse scene flow dataset. The total grid is of size 160×150.

| Grid cell size (m) | $\mathcal{E}\downarrow$ $(m)$ | $Acc_5\uparrow$ (%) | $Acc_{10}\uparrow$ (%) | $\theta_\epsilon\downarrow$ $(rad)$ | t$\downarrow$ $(ms)$ | Mem.$\downarrow$ $(GB)$ |
|---|---|---|---|---|---|---|
| 1 | 0.225 | 4.30 | 19.91 | 0.189 | 342 | 1.33 |
| 0.5 | 0.123 | 34.69 | 70.98 | 0.135 | 348 | 1.33 |
| 0.33 | 0.089 | 73.25 | 92.51 | 0.116 | 353 | 1.35 |
| 0.2 | 0.071 | 90.15 | 95.36 | 0.105 | 419 | 1.35 |
| 0.1 | 0.060 | 94.31 | 95.35 | 0.097 | 419 | 1.37 |
| 0.05 | 0.059 | 94.22 | 95.31 | 0.097 | 579 | 1.51 |
| 0.02 | 0.060 | 94.26 | 95.46 | 0.095 | 1151 | 2.68 |
| 0.01 | 0.058 | 93.92 | 95.25 | 0.095 | 1438 | 7.42 |

gap between the learning methods and the non-learning-based methods lies in the OOD generalizability. Even trained on similar lidar sensors—*e.g.*, FlowStep3D was trained on the KITTI [6] dataset—the Waymo Open [11] and Argoverse [2] scene flow datasets have different point cloud range, coordinate configurations, *etc*. to KITTI dataset, making the pre-trained model vulnerable to these data variations. In contrast, non-learning-based methods maintained high accuracy on different datasets. Note that our method still has competitive efficiency among these learning methods.

### 1.6. Additional results of DT grid size

We provide additional results on a 2D bird's eye view (BEV) scene in Tab. 3. The result is aligned with the main paper Fig. 6.

### 1.7. 2D BEV visual results

Some visual results of the 2D BEV scenes of the Argoverse scene flow dataset are shown in Fig. 1.

### 1.8. Visual results

Please see Fig. 2 and the project webpage https://lilac-lee.github.io/FastNSF for more visual results and applications.

## 2. Application: point accumulation

The implicit and continuous neural function allows for easy point accumulation with per-pair scene flow estimation. Moreover, with the speedup that our method has achieved, the computation of point accumulation substantially decreased, making it possible for large amounts of point densification.

### 2.1. Continuous scene flow field

It is important to note that using DT to replace CD will not alter the continuous property. Therefore, similar to NSFP, our method creates a continuous flow field in that the network itself interpolates the motion of the entire 3D space, enabling long-term flow estimation and point densification through forward integration.

### 2.2. Dense point cloud accumulation

We followed [5] to accumulate point clouds using Euler integration with per-pair scene flow estimation for the Argoverse scene flow dataset. Different from [5], we compute per-pair scene flow for each consecutive pair (*i.e.*, frame 1→2, frame 2→3, ..., frame 10→11) and interpolate fast neural scene flow to integrate 10 point clouds following the reference frame into the reference frame to densify the depth map and the point cloud. Some visual examples are shown in Fig. 3.

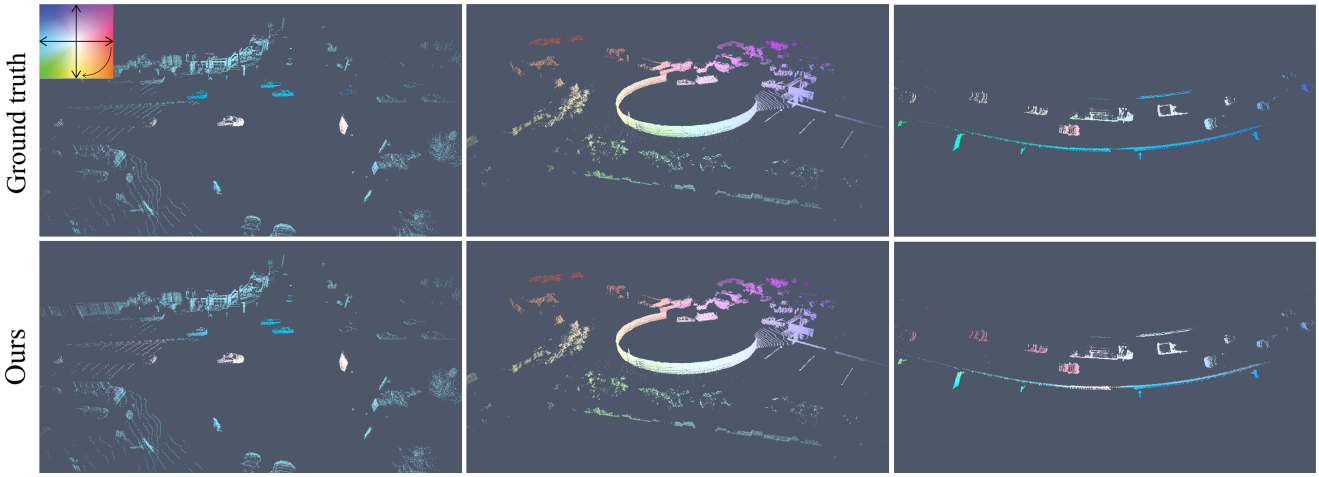
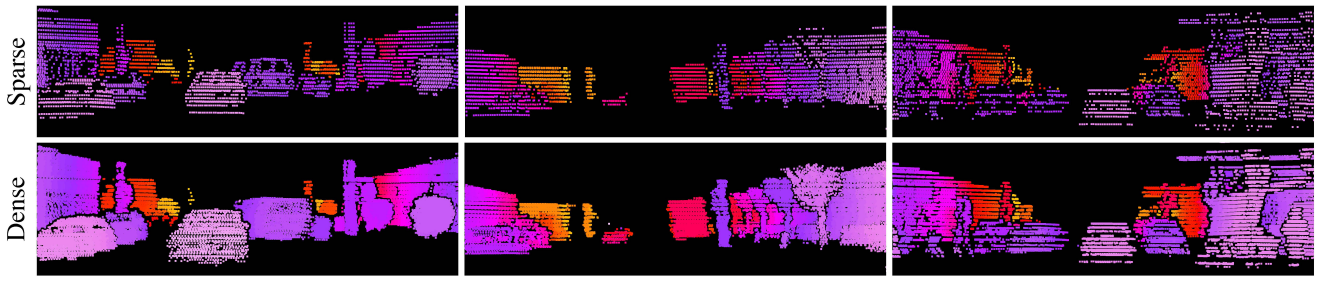
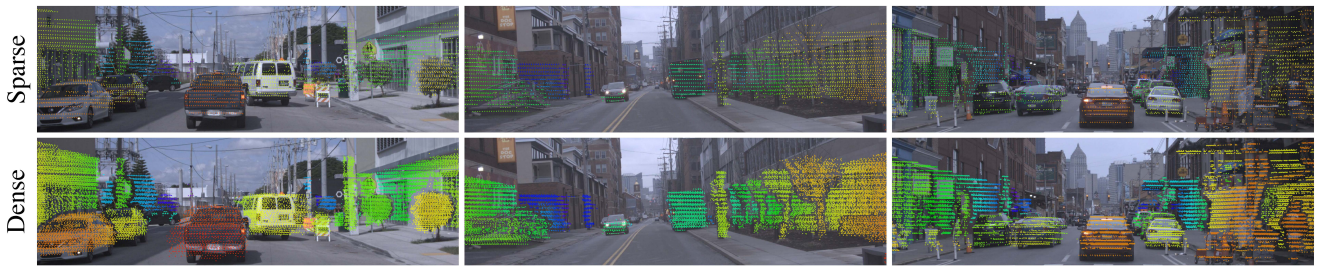



Figure 2. Visual examples of the scene flow prediction using our method on Waymo Open scene flow dataset.




(a) Lidar depth map densification




(b) Lidar point cloud accumulation

Figure 3. Visualization of the application of our method: (a) depth densification, and (b) point cloud accumulation. We present the original sparse scenes on the upper row and the densified results on the lower row. For point cloud accumulation, we projected the densified point cloud to the corresponding image plane for better visualization.

## References


[1] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018. 2

[2] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3D tracking and forecasting with rich maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8748–8757, 2019. 4

[3] Zan Gojcic, Or Litany, Andreas Wieser, Leonidas J Guibas, and Tolga Birdal. Weakly supervised learning of rigid 3d scene flow. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5692–5703, 2021. 3

[4] Yair Kittenplon, Yonina C Eldar, and Dan Raviv. FlowStep3D: Model unrolling for self-supervised scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3

[5] Xueqian Li, Jhony Kaesemodel Pontes, and Simon Lucey. Neural scene flow prior. *Advances in Neural Information Processing Systems*, 34, 2021. 3, 4

[6] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3061–3070, 2015. 4

[7] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 405–421. Springer, 2020. 1

[8] Jhony Kaesemodel Pontes, James Hays, and Simon Lucey. Scene flow from point clouds with or without learning. In *Proceedings of the International Conference on 3D Vision (3DV)*. IEEE, 2020. 3

[9] Gilles Puy, Alexandre Boulch, and Renaud Marlet. FLOT: Scene Flow on Point Clouds Guided by Optimal Transport. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 3

[10] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. 2

[11] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 4

[12] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Neural Information Processing Systems (NeurIPS)*, 2020. 1

[13] Yi Wei, Ziyi Wang, Yongming Rao, Jiwen Lu, and Jie Zhou. Pv-raft: point-voxel correlation fields for scene flow estimation of point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6954–6963, 2021. 3

[14] Wenxuan Wu, Zhi Yuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. PointPWC-Net: Cost volume on point clouds for (self-) supervised scene flow estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 88–107. Springer, 2020. 3

[15] Jianqiao Zheng, Sameera Ramasinghe, Xueqian Li, and Simon Lucey. Trading positional complexity vs deepness in coordinate networks. In *European Conference on Computer Vision*, pages 144–160. Springer, 2022. 1, 2