Supplementary Materials for MonoPLFlowNet: Permutohedral Lattice FlowNet for Real-Scale 3D Scene Flow Estimation with Monocular Images

Runfa Li and Truong Nguyen

UC San Diego
{rul002, tqn001}@ucsd.edu

The supplementary materials are referred at several places in the paper, and the corresponding sections are organized in the same order as mentioned in the paper. [1]

1 Permutohedral Lattice Network Operations

Our scene decoder architecture design is based on the Hierarchical Permutohedral Lattice Network (HPL) [4], which is designed on top of Bilateral Convolutional Layers (BCL) [7]. BCL derives the general operation for processing features in lattice: "Splatting-Convolution-Slicing", as we mentioned in Section 3.1 of the paper.

Splatting: After the signals are embedded to the lattice, BCL splats the signal values to the enclosing lattice vertices by barycentric interpolation, and stores barycentric weights at each vertex in a hash table. The splatting complexity is in $O(d^2)$ time, which is more efficient than splatting in a traditional Cartesian coordinate.

Convolution: BCL imitates CNN convolution and designs the permutohedral lattice convolution the same way. [7] shows the BCL convolution on a 2D permutohedral lattice with a Gaussian kernel sliding over the lattice, and further improves it to non-separable filter kernels in learnable manner.

Slicing: With the barycentric weights stored in splatting process, the convolved signals (features) can be sliced back to the original position. Another advantage of slicing in permutohedral lattice is that instead of the original position, the signals can be sliced to different positions, for example a scaled lattice.

Interested readers should refer to Figure 1 of [7] for the visual illustration of "Splatting-Convolution-Slicing". For more properties of the permutohedral lattice, please refer to [1, 7, 4].

[1] Table, Equation and Figure in red fonts refer to those in the supplementary materials, while Table, Equation and Figure in red fonts refer to those in the original paper.

2 MonoPLFlowNet Scene Decoder Details

Both monocular-image and LiDAR based approaches mask out the occluded and invalid pixels for training and evaluation. However, different to monocular-image based approaches using all pixels after masking [5,6], the LiDAR based approaches randomly select a specific number N of 3D points. For the estimation in 3D, we follow the same strategy as of the LiDAR based approaches [8,12,4,10], and randomly choose $N = 8,192$ pixels from the two consecutive input images.

Following Eq. 9 in the paper, we first project the 8,192 randomly chosen pixels from the estimated depth map to 3D points, and then project 3D points to permutohedral lattice. With these 8,192 points in the lattice, we build the pyramid-level lattice layers with scale 1, 0.5 and 0.25 corresponding to level 1, 2 and 4 in Figure 2 of the paper. In the figure, N_2, N_3, N_4 are the number of activated lattice vertices corresponding to level 1, 2 and 4. This is similar to the Hierarchical design of HPL [4]. The lattice with smaller scale (deeper level) have larger receptive field in each cell. This is also similar to traditional CNN in Euclidean grid, which means that with larger receptive field in lattice cell, features need fewer lattice vertices to store their barycentric weights, i.e., $N_4 < N_3 < N_2 < N_1 = 8,192$. In Eq. 9 of the paper, we show that scaling the feature position in permutohedral lattice leads to a same scale to the feature depth map at 2D Cartesian coordinate. Since our specific-designed depth decoder also estimates the depth at scale 1, 0.5 and 0.25 (last three levels), we label the 2D coordinates of the 8,192 pixels in the depth map corresponding to 8,192 points in the lattice, and derive the corresponding 2D coordinates of n_2, n_3, n_4 number of pixels at scale 1, 0.5 and 0.25 by bilinear interpolation, so that $n_1 = 8,192 \approx 4n_2 \approx 16n_3 \approx 64n_4$. Then we splat and slice the high-dimensional features at the 2D coordinates of the depth map to the corresponding level of permutohedral lattice, namely splat and slice n_4 features from depth map to N_4 lattice vertices, n_3 to N_3 and n_2 to N_2. Note that $n_1 = N_1 = 8,192$ does not mean that n_2, n_3, n_4 and N_2, N_3, N_4 are exactly equal. With more features from different depth levels concatenated to different lattice levels, the performance improves as shown in the ablation study (Table 5 of the paper).

3 Evaluation Details

LiDAR-based Evaluation Standard: In Section 4.2 of the paper, we show the Image-based Evaluation Standard, and use it for the comparison of monocular-image based approaches in Tables 3, 4 and 5 of the paper. Here we show the original LiDAR-based Evaluation Standard [4,2], which is used for Table 6 of the paper.

- EPE3D(m): 3D end point error averaged over each point in meters.
- Acc3DS: the percentage of points with EPE3D < 0.05m or relative error $<$ 5%.

Dataset	Method	Input	Scale	Size	Device	Runtime (ms)
KITTI	EPC[15]	Mono-RGB	✗	-	-	50
	EPC++[9]					50
	Mono-SF-old[3]					41000
	Mono-SF[5]			image = (256, 832)		90
	Multi-Mono-SF[6]					63
	Ours-depth		✓	image = (352, 1216) points = 8192	1080 Ti	73
	Ours-lattice					183
	Ours-scene					6
	Ours					≈ 262
Flyingthings3D	FlowNet3D[8]	LiDAR	✓	points = 8192	Titan V	130.8
	HPLFlowNet[4]					98.4
	PointPWC-net[13]				1080Ti	117.4
	Self-HPLFlowNet[11]				-	491.3
	Ours-depth	Mono-RGB		image = (544, 960) points = 8192	1080 Ti	65
	Ours-lattice					103
	Ours-scene					11
	Ours					≈ 179

Table 1. Inference Speed Comparison. ✓denotes in real scale, ✗ denotes with scale ambiguity.

- Acc3DR: the percentage of points with EPE3D < 0.1m or relative error $<$ 10%.
- Outliers3D: the percentage of points with EPE3D > 0.3m or relative error $>$ 10%.
- EPE2D(px): 2D end point error obtained by projecting point clouds back to the image plane.
- Acc2D: the percentage of points whose EPE2D < 3px or relative error $< 5\%$.

Scale and 3D Scene Flow Recovery: In Section 4.2 of the paper, in order to evaluate other monocular-image based approaches, we need to recover the depth and scene flow scale. There are two cases:

Case 1. Since approaches in [5, 6] directly estimate 3D scene flow vectors, then we need to recover the scale of depth and scene flow. To recover the depth scale:

$$\begin{aligned} S_i &= \bar{d}_i / \bar{\tilde{d}}_i \\ d_i &= S_i \tilde{d}_i \end{aligned} \tag{1}$$

where d_i and $\tilde{d}_i$ are the ground truth and the estimated depth map at frame i. S is used to recover to real scale which is the ratio between the average ground truth at that frame $\bar{d}_i$ to the average estimated depth $\bar{\tilde{d}}_i$. For scene flow recovery, we just need to recover the scale similarly:

$$\begin{aligned} S_i &= s\bar{f}_{zi} / s\bar{\tilde{f}}_{zi} \\ sf_i &= S_i s\tilde{f}_i \end{aligned} \tag{2}$$

where sf_i and $s\tilde{f}_i$ are the ground truth and the estimated 3D scene flow at frame i. S is used to recover to real scale which is the ratio between the average ground truth of the scene flow in Z axis (depth dimension) at that frame $s\bar{f}_{zi}$ to the average estimated one $s\bar{\bar{f}}_{zi}$.

Case 2. Since other work [14] only estimates 2D optical flow, we need to use the depth (Eq. 1) and 2D optical flow to recover the 3D scene flow:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (z+e_z)/f_u & 0 & (-c_u z + e_x)/f_u \\ 0 & (z+e_z)/f_v & (-c_u z + e_y)/f_v \\ 0 & 0 & z \end{bmatrix} \begin{bmatrix} u + sf_x \\ v + sf_y \\ 1 \end{bmatrix} \tag{3}$$

$$sf_{3d} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4}$$

where (sf_x, sf_y) is the 2D optical flow, (e_x, e_y, e_z) are camera extrinsic parameters, (x, y, z) are 3D point in the first frame projected from the 2D image pixel (u, v), defined by Eq. 8 in the paper. (x, y, z) is the end 3D point corresponding to (x, y, z). The rest of notations are consistent to Eq. 8 of the paper. Note that Eq. 8 is the simplified version of Eq. 3 where the 2D optical flow and camera extrinsic parameters are set to zero. Camera extrinsic parameters are not considered for Flyingthings3D dataset, but considered for KITTI dataset because there is a relative transformation between the camera and LiDAR when data is collected. The parameters of the transformation is available in the dataset.

Since [14] only estimates 2D optical flow without depth estimation, we use Mono-SF [5] depth (recovered to real scale by Eq. 1) and our depth to recover the 3D scene flow, and scale it with ground truth by Eq. 2, as shown in Table 4 of the paper. For 3D scene flow evaluation on KITTI flow 2015, to be consistent to the evaluations in other LiDAR-based works [4, 13], we evaluate on the same 142 frames of the complete 200 frames.

Domain Generalization: Since we use fully supervised training with ground truth 3D scene flow labels, we only train on synthetic dataset Flyingthings3D and use it directly on real dataset KITTI. Such a domain generalization strategy is widely used in LiDAR based works [8, 4, 13]. However, these LiDAR based approaches achieve better performance on KITTI than ours, because scene flow ground truth is generated in different ways for different datasets.

In LiDAR approaches [4, 2], point cloud of the first frame is generated in same way for both datasets by using first frame depth ground truth, as shown in Eq. 8. For Flyingthings3D dataset, the point cloud of second frame is generated by the first frame depth and 2D optical flow ground truth - strategy A, as in Eq. 3. While for KITTI dataset, the second frame's point cloud is generated by using second frame's ground truth depth - strategy B, as in Eq. 8. Then the scene flow ground truth are generated as the translation between the two point clouds.

Such difference does not have negative impact on LiDAR based approaches, because in evaluation the input to the model is still point cloud. However, it has more negative impact on our image-based model, because with strategy A, the model is trained to estimate the scene flow that is not associating to the second image but the point cloud translated by 2D optical flow ground truth. To improve our domain generalization from synthetic to real dataset, we will train on Flyingthings3D with strategy B for future work.

Inference Speed: Table 1 summarizes the inference speed of both LiDAR and monocular-image based works and our work on both KITTI flow 2015 and Flyingthings3D evaluation datasets. From the table, it is clear that monocular-image based approaches are faster than LiDAR based approaches, at the expense of accuracy and real scale. Our total run time simply consists of three major parts, depth decoder, scene flow decoder, and the process of generating permutohedral lattice. Although we estimate the depth, the depth decoder only needs to run once for consecutive frames, where features from the depth decoder at current frame can be stored for the next frame pairs to estimate scene flow, so the total run time is approximately the overall time over these three parts. Using GTX 1080Ti without further optimizing the code, our inference speed shows the potential to be used for real-time application.

4 Visual Results

Demo Video: We include two demo videos on KITTI and Flyingthings3D respectively, please check the supplementary materials submission.

Original RGB Input: In Figure 5 of the paper, the last three columns show the visual results on Flyingthings3D dataset. We show the corresponding RGB image (first frame) input as Figure 1.

Qualitative visual results: We show visual comparison of our depth and 3D scene flow estimation to the two strongest baseline works Mono-SF[5] and Mono-SF-Multi[6], where Figures 2 to 5 are on Flyingthings3D, and Figures 6 to 9 are on KITTI. For 3D scene flow visualization, blue points are from frame t, red and green points are blue points translated to frame $t+1$ by ground truth and estimated 3D scene flow, respectively.

Failure Cases: Figure 10 shows a representative failure case of our methodology. Outside of the white box, our 3D scene flow estimation still shows good performance, but inside the white box, the estimated flow is not enough to compensate the motion. This is a typical problem for most scene flow works, where rigid-object motions have larger error. In Figure 10, the white box shows a vehicle moving in the opposite direction to the ego motion, which has large relative motion to the background. While the vehicles align at the right side are static to the background, which have zero relative motion to the background, this is

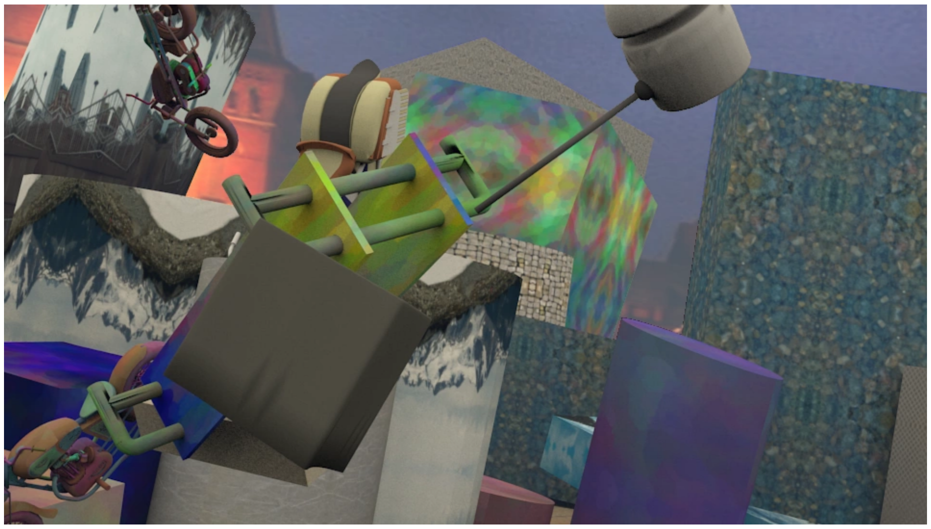

Fig. 1. Corresponding RGB image input to the paper Figure. 5 Flyingthings3D.

the reason that the vehicle inside of the white box has large error. In our future work, we will study rigid and non-rigid motion separately to further boost the performance.

5 Limitation and Future Works

Besides the common failure case described above, our current model has the limitation that the depth/3D scene flow estimation has to be trained to test in the similar domain, for the model tested in different domain to the training set, it fails to estimate in high accuracy. For a different dataset, the model requires enough labels from the domain to train. Another limitation is the performance gap of our monocular image-based model to the SOTA LiDAR-based models.

For future works, we aim to strengthen the generalization ability of the model, and explore self-supervised training to overcome the lack of data. We will also further improve the accuracy to be competitive to SOTA LiDAR-based work and enhance the efficiency for real-time application.

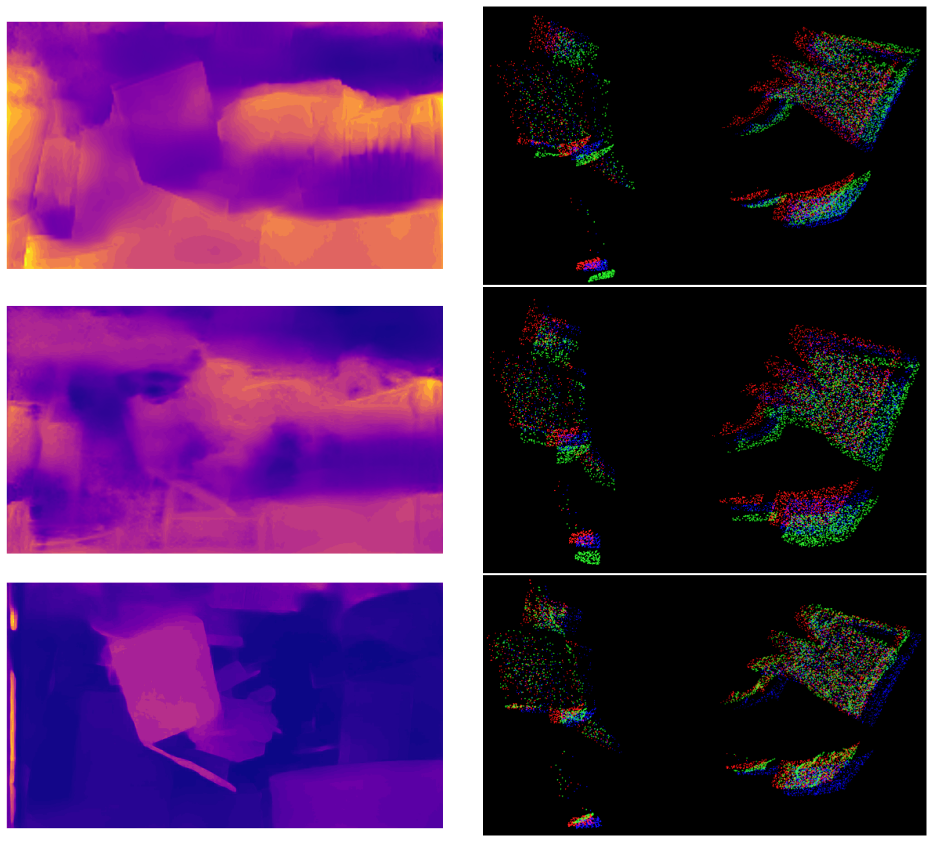
Fig. 2. Comparison on Flyingthings3D. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.2979 m**.

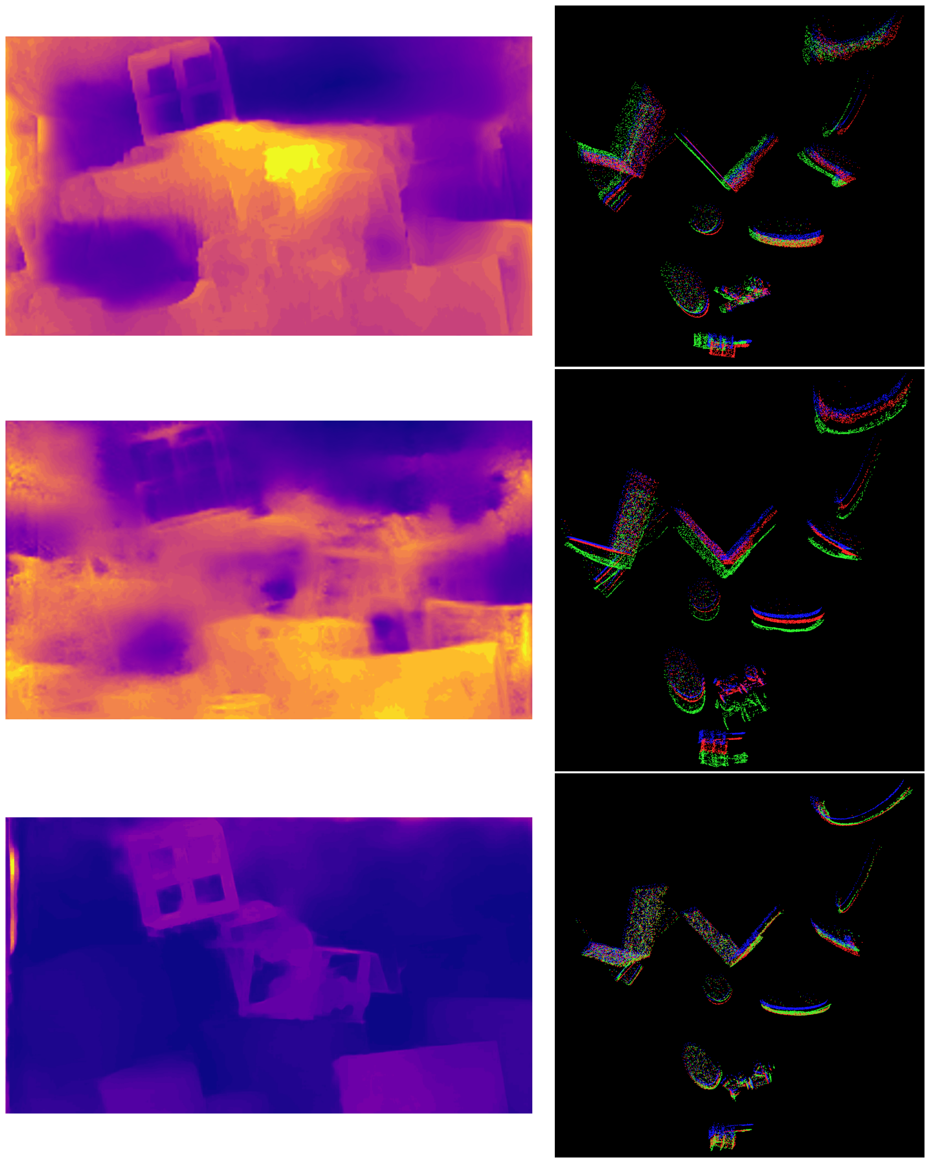

Fig. 3. Comparison on Flyingthings3D. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.1343 m**.

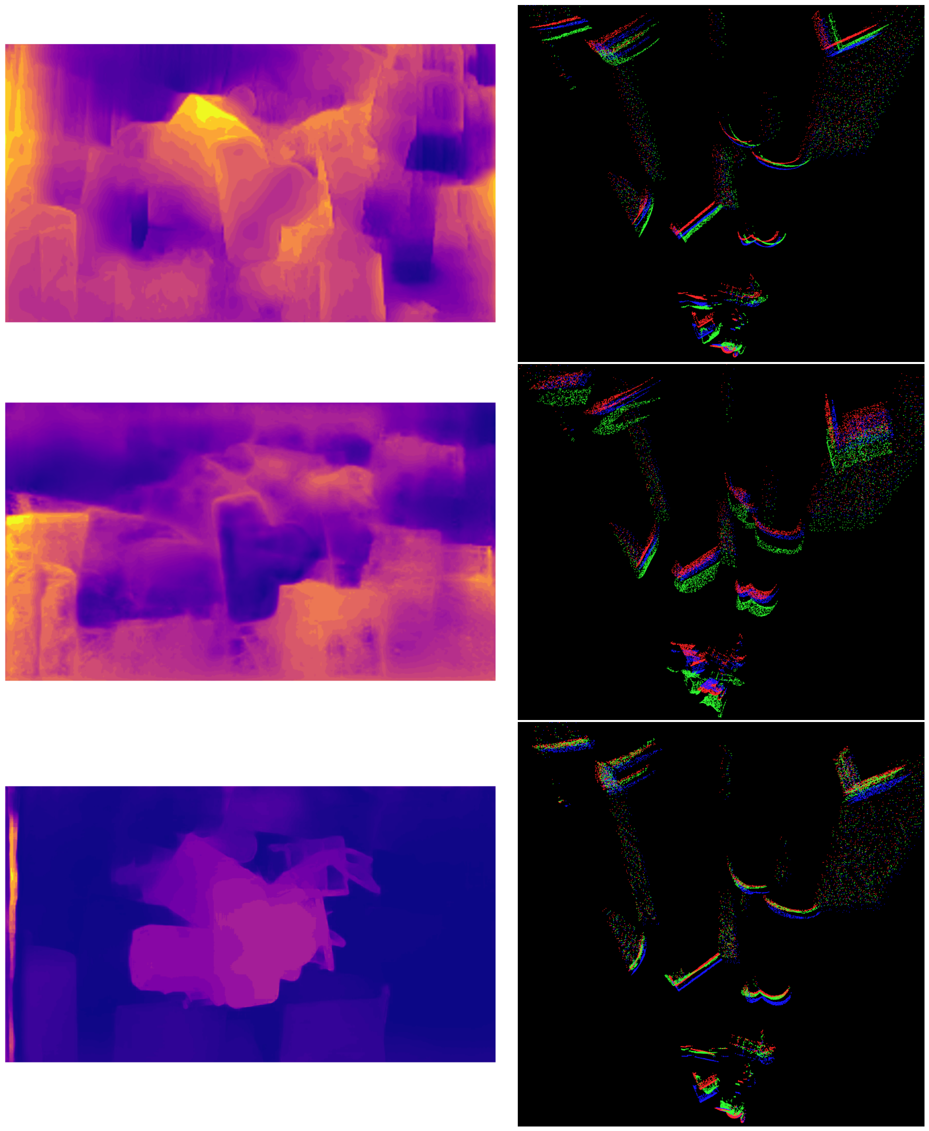

Fig. 4. Comparison on Flyingthings3D. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.1399 m**.

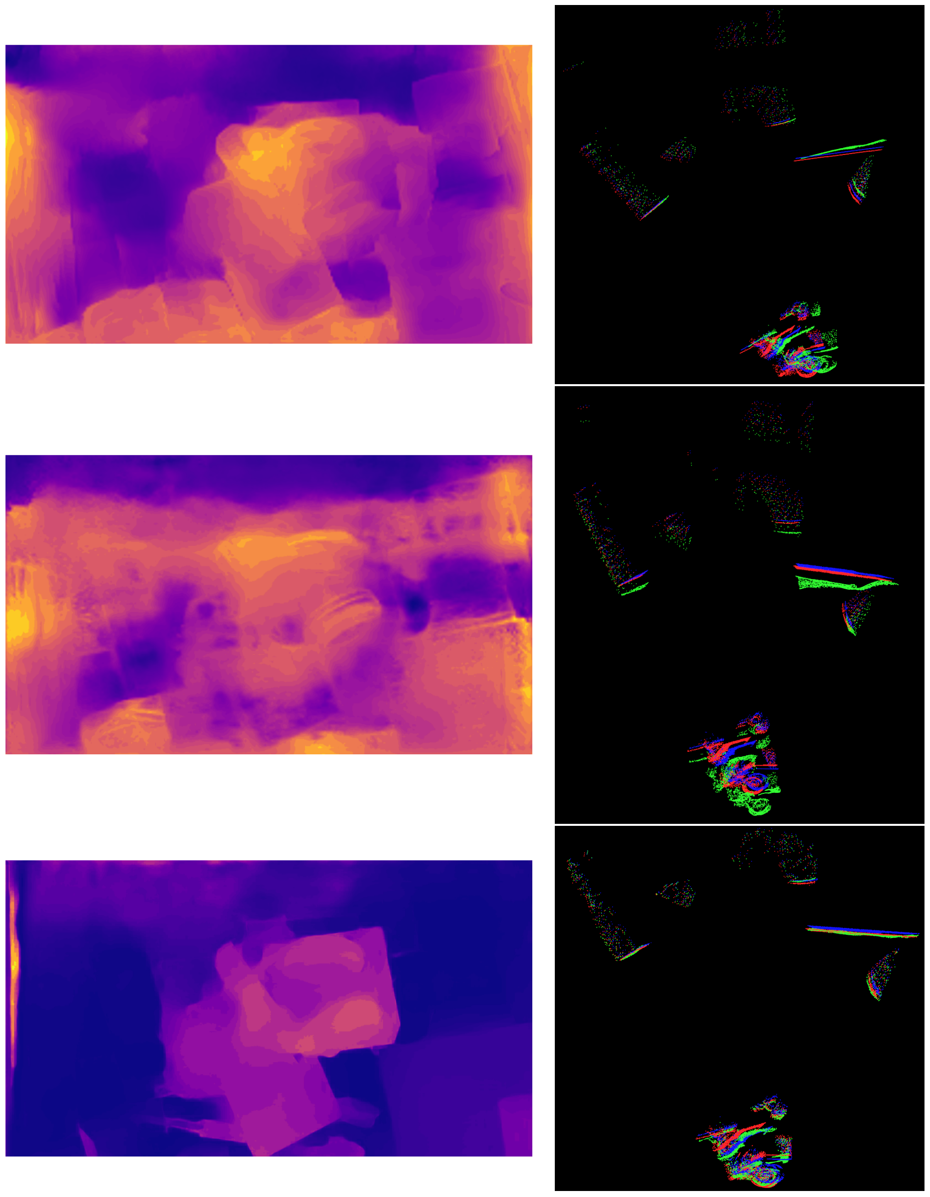
Fig. 5. Comparison on Flyingthings3D. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.1212 m**.

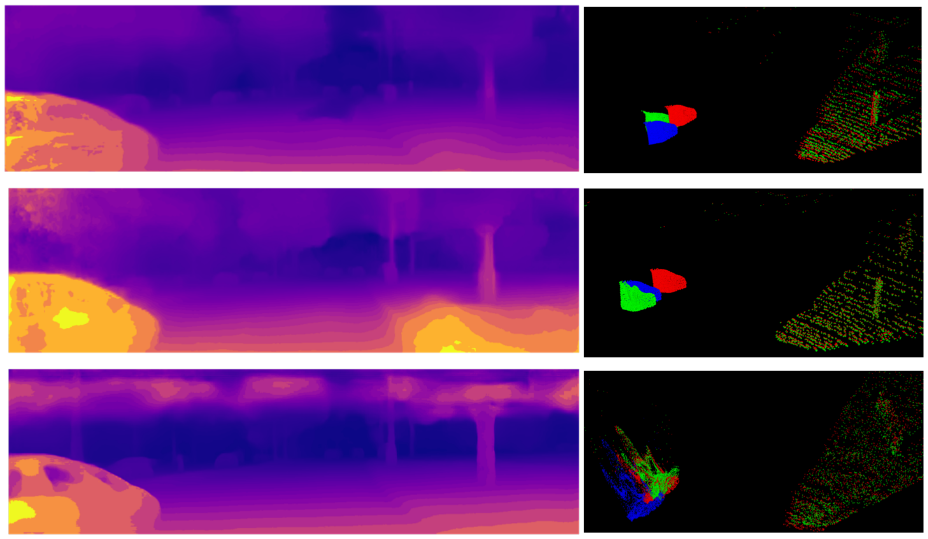

Fig. 6. Comparison on KITTI. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.4010 m**.

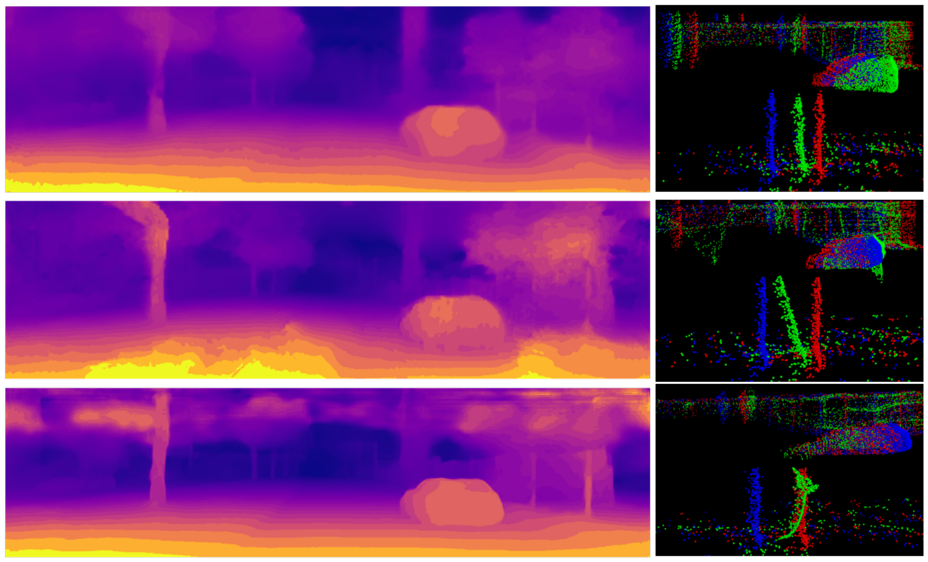

Fig. 7. Comparison on KITTI. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.3909 m**.

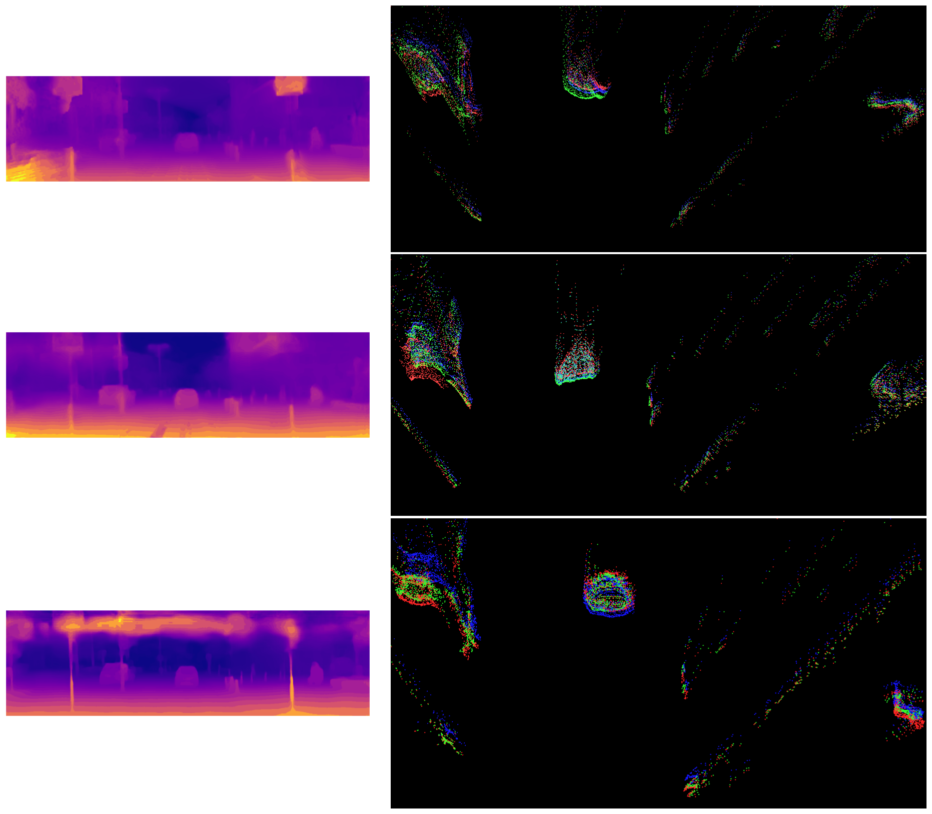

Fig. 8. Comparison on KITTI. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.3022 m**.

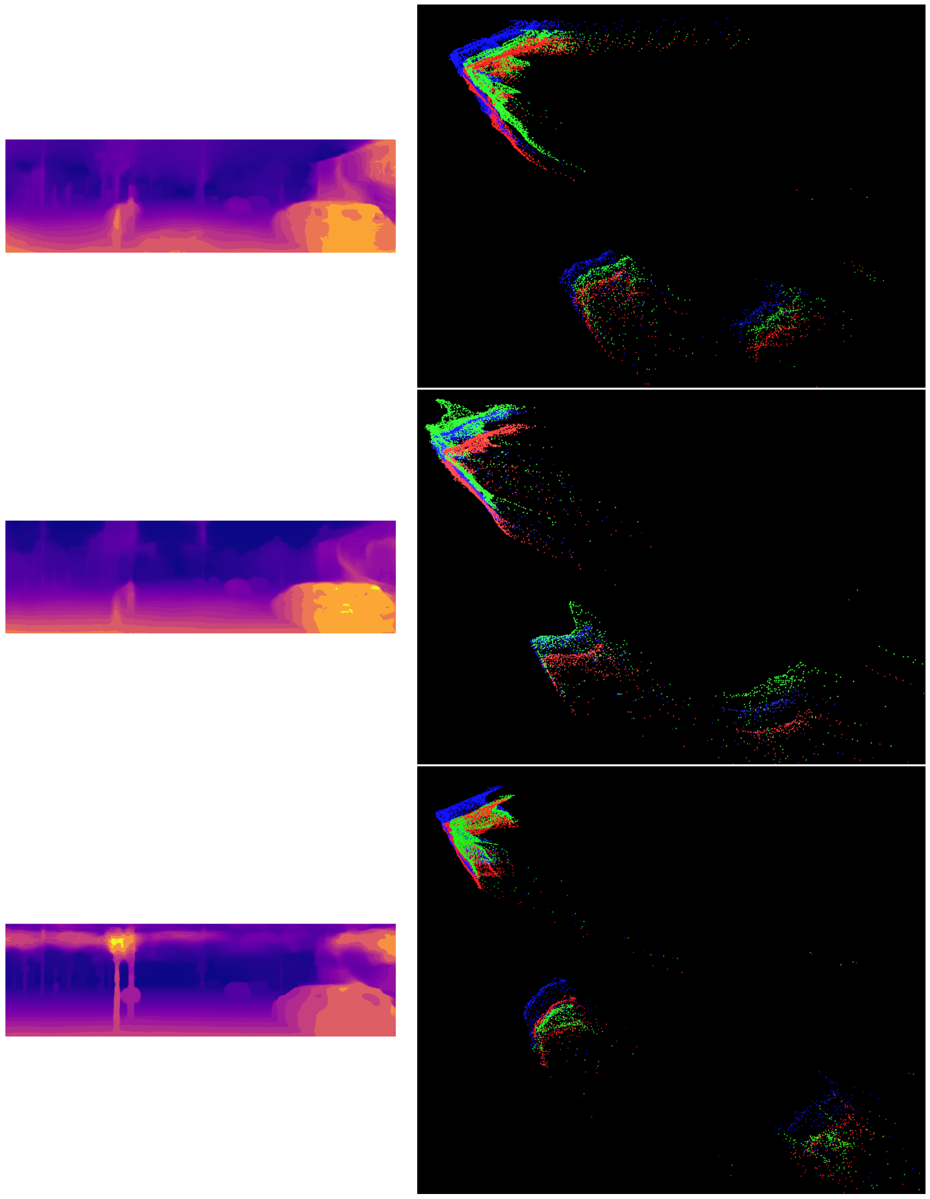

Fig. 9. Comparison on KITTI. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.3168 m**.

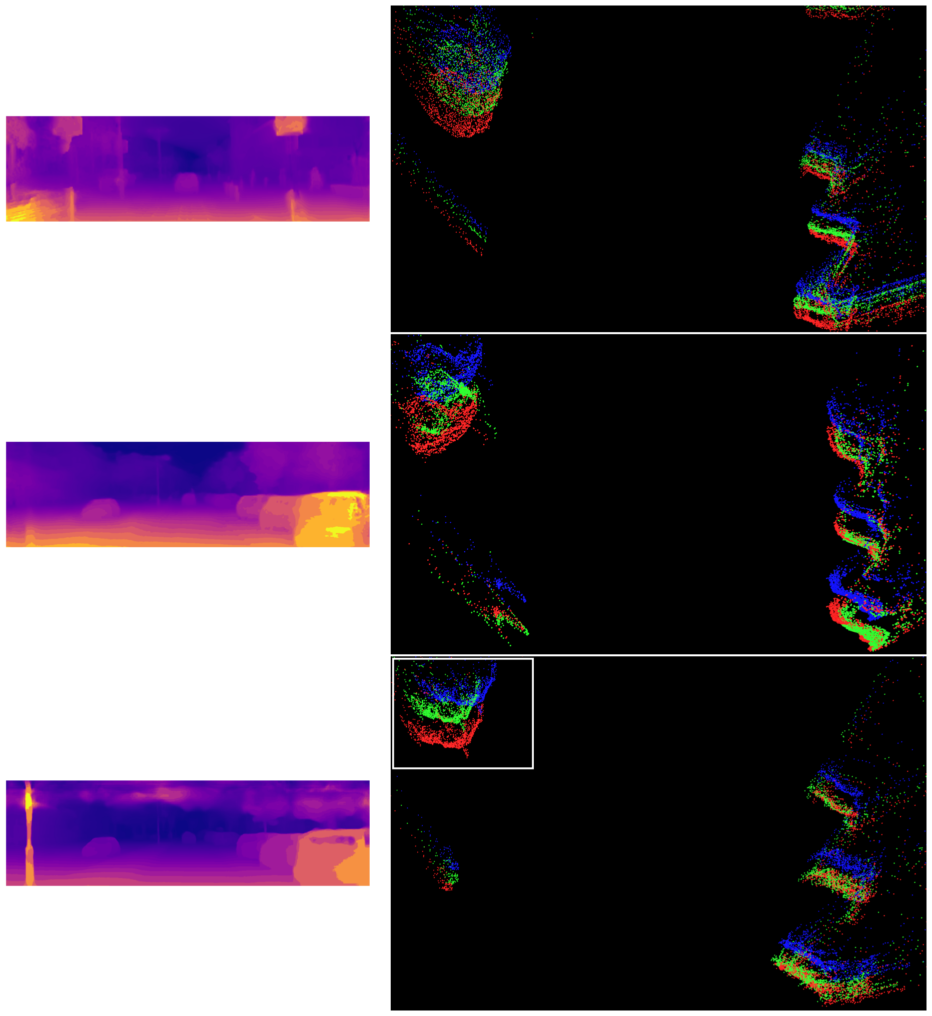

Fig. 10. Failure case on KITTI. 1st column: depth estimation. 2nd column: 3D scene flow estmation. 1st row: Mono-SF[5]. 2nd row: Mono-SF-Multi[6]. 3rd row: Ours, **EPE3D = 0.3326 m**.White box shows the failure case of our 3D scene flow estimation on the non-rigid motion, inside the box is a moving vehicle with high relative speed to the background.

References

1. Adams, A., Baek, J., Davis, M.A.: Fast high-dimensional filtering using the permutohedral lattice. Computer Graphics Forum **29** (2010)
2. Behl, A., Paschalidou, D., Donné, S., Geiger, A.: Pointflownet: Learning representations for rigid motion estimation from point clouds. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7954–7963 (2019). https://doi.org/10.1109/CVPR.2019.00815
3. Brickwedde, F., Abraham, S., Mester, R.: Mono-sf: Multi-view geometry meets single-view depth for monocular scene flow estimation of dynamic traffic scenes. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 2780–2790 (2019). https://doi.org/10.1109/ICCV.2019.00287
4. Gu, X., Wang, Y., Wu, C., Lee, Y.J., Wang, P.: Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3249–3258 (2019). https://doi.org/10.1109/CVPR.2019.00337
5. Hur, J., Roth, S.: Self-supervised monocular scene flow estimation. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7394–7403 (2020). https://doi.org/10.1109/CVPR42600.2020.00742
6. Hur, J., Roth, S.: Self-supervised multi-frame monocular scene flow. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2684–2694 (June 2021)
7. Jampani, V., Kiefel, M., Gehler, P.V.: Learning sparse high dimensional filters: Image filtering, dense crfs and bilateral neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4452–4461 (2016). https://doi.org/10.1109/CVPR.2016.482
8. Liu, X., Qi, C.R., Guibas, L.J.: Flownet3d: Learning scene flow in 3d point clouds. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 529–537 (2019). https://doi.org/10.1109/CVPR.2019.00062
9. Luo, C., Yang, Z., Wang, P., Wang, Y., Xu, W., Nevatia, R., Yuille, A.: Every pixel counts ++: Joint learning of geometry and motion with 3d holistic understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence **42**(10), 2624–2641 (2020). https://doi.org/10.1109/TPAMI.2019.2930258
10. Sun, D., Yang, X., Liu, M.Y., Kautz, J.: Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8934–8943 (2018). https://doi.org/10.1109/CVPR.2018.00931
11. Tishchenko, I., Lombardi, S., Oswald, M.R., Pollefeys, M.: Self-supervised learning of non-rigid residual flow and ego-motion. In: 2020 International Conference on 3D Vision (3DV). pp. 150–159 (2020). https://doi.org/10.1109/3DV50981.2020.00025
12. Wang, Z., Li, S., Howard-Jenkins, H., Prisacariu, V.A., Chen, M.: Flownet3d++: Geometric losses for deep scene flow estimation. In: 2020 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 91–98 (2020). https://doi.org/10.1109/WACV45572.2020.9093302
13. Wu, W., Wang, Z.Y., Li, Z., Liu, W., Fuxin, L.: Pointpwc-net: Cost volume on point clouds for (self-) supervised scene flow estimation. In: European Conference on Computer Vision. pp. 88–107. Springer (2020)
14. Yang, G., Ramanan, D.: Upgrading optical flow to 3d scene flow through optical expansion. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1331–1340 (2020). https://doi.org/10.1109/CVPR42600.2020.00141

15. Yang, Z., Wang, P., Wang, Y., Xu, W., Nevatia, R.: Every pixel counts: Unsupervised geometry learning with holistic 3d motion understanding (2018)