Supplementary Material for CamLiFlow

Haisong Liu Tao Lu Yihui Xu Jia Liu Wenjie Li Lijun Chen
State Key Laboratory for Novel Software Technology, Nanjing University, China
{liuhs, taolu, yhxu, wenjielee}@smail.nju.edu.cn, {jialiu, chenlj}@nju.edu.cn

In this supplementary material, we first present our proposed strategies to improve training stability. Next, we introduce the implementation details of CamLiFlow. Finally, we provide additional qualitative examples on FlyingThings3D and KITTI.

1. CamLiFlow

1.1. Improving Training Stability

Gradient Detaching. Fusing two branches belonging to different modalities may encounter scale-mismatched gradients, making the training unstable and dominated by one modality. To solve the issue, we propose to detach the gradient from one branch to the other for each Bi-CLFM, as shown in Fig. 4 in the main paper. In Fig. 10, we conduct an ablation to demonstrate the effect of gradient detaching. Without gradient detaching, the image branch dominates the training and hurts the performance of the point branch. By detaching the gradient from one branch to the other, we prevent one modality from dominating so that each branch focuses on its task.

Batch Normalization. Although the original implementation of PWC-Net does not use batch normalization (BN) [4] in the whole network, we empirically find that integrating BN into the feature pyramid can improve training stability and speed up the convergence.

1.2. Implementation Details

Feature Pyramid. As shown in Fig. 11, we generate a feature pyramid for the image branch and the point branch respectively, with the top-level being the inputs. Our feature pyramid follows the implementation of the original PWC-Net [7] and PointPWC-Net [9] but with some modifications. For the image branch, we replace the feedforward CNNs with residual blocks [1] and perform batch normalization for each convolutional layer. For the point branch, the original PointPWC-Net builds a three-level pyramid and downsamples the points by a factor of 4, while we build a six-level pyramid with a downsampling factor of 2 to match the levels of the image branch. Features from level 6 to level 2 are fused by a Bi-CLFM.

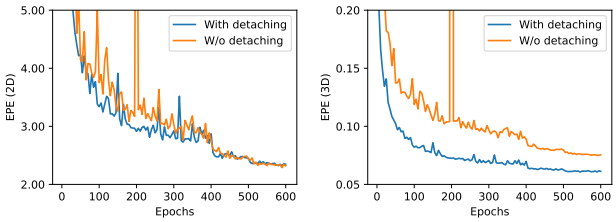


Figure 10. Performance on the "val" split of FlyingThings3D with/without our gradient detaching strategy. Detaching the gradient from one branch to the other can prevent one modality from dominating, so that each branch focuses on its own task.

We start processing from the top level and perform the coarse-to-fine estimation scheme until level 2. Hence, CamLiFlow outputs optical flow at 1/4 resolution and scene flow at 1/2 resolution. We obtain full-scale optical flow and scene flow by convex upsampling [8] and k-NN upsampling respectively.

Warping. For image features, warping can be directly implemented with *bilinear grid sampling*. For point clouds, however, grid sampling no longer works since points do not conform to the regular grids as in images. In PointPWC-Net, the warping layer is simply an element-wise addition between the scene flow and the point clouds of the first frame. However, this does not work in our case, since PointPWC-Net warps the first frame towards the second frame, while our image branch warps the second frame towards the first frame. This mismatch can make it difficult for training to converge.

Hence, we warp the point clouds of the second frame towards the first frame using the backward scene flow, which can be computed using *inverse distance weighted interpolation*. Formally, let P_1^l and P_2^l be the point clouds of the first frame and the second frame at level l respectively. We first warp P_1^l to the target frame using the upsampled scene flow from level $l+1$:

$$\tilde{P}_1^l = P_1^l + \mathbf{U}(f_{3D}^{l+1}), \tag{1}$$

where $\mathbf{U}$ denotes the upsampling operator based on k-NN interpolation and f_{3D}^{l+1} denotes the coarse scene flow from

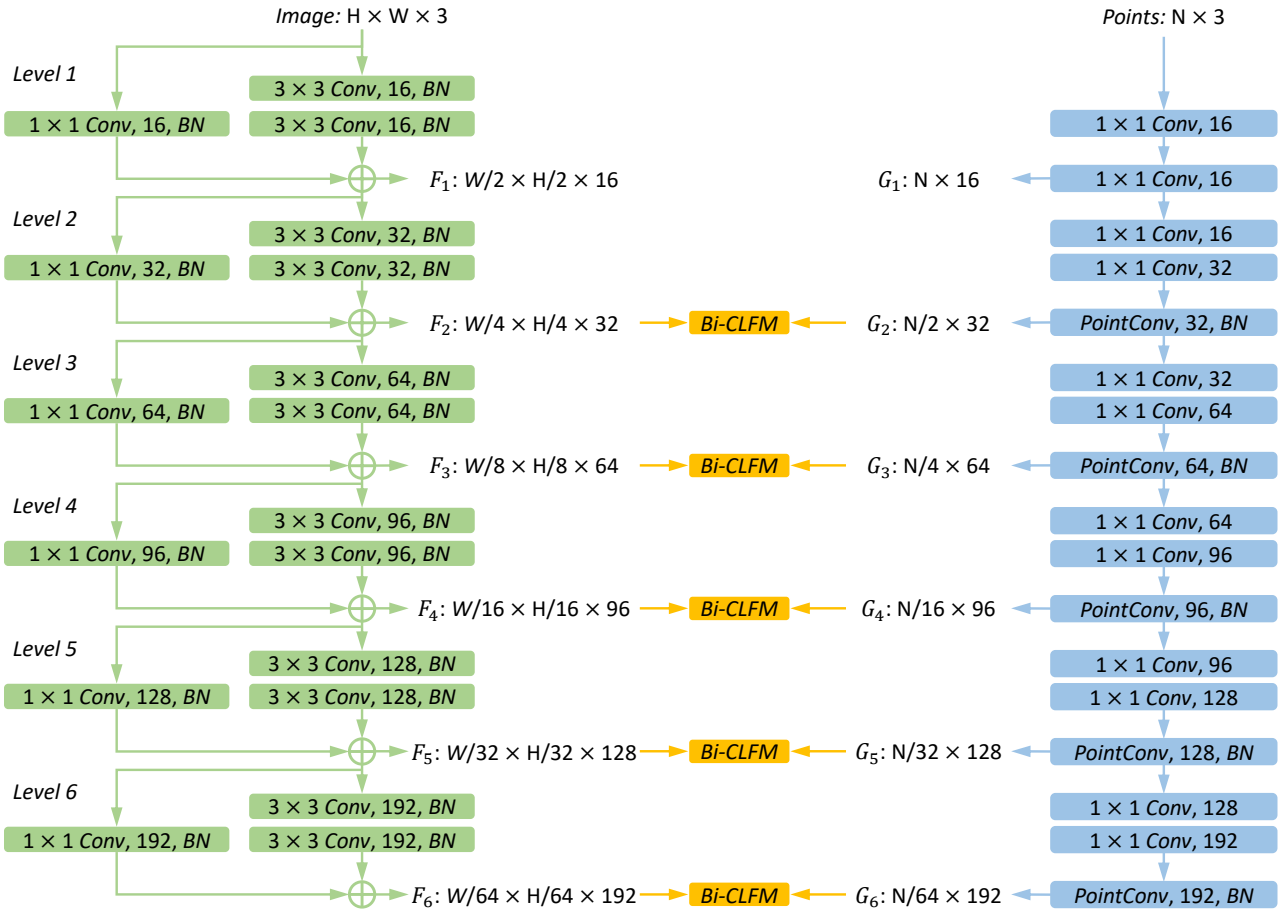


Figure 11. The feature pyramid of CamLiFlow. We build a feature pyramid for the image branch and the point branch respectively, with the top-level being the inputs. For the image branch, we extract features with residual blocks and perform batch normalization for each convolution layer. For the point branch, we build a six-level pyramid with a downsampling factor of 2 to match the levels of the image branch. Features from level 6 to level 2 are fused by a Bi-CLFM to pass complementary information.

level $l+1$. Then, for each point in P_2^l, we find its k nearest neighbor (k-NN) in $\tilde{P}_1^l$. The backward flow b_{3D}^l can be computed using inverse distance weighted interpolation:

$$b_{3D}^l(i) = -\frac{\sum_{j=1}^k w_{ij} f_{3D}^{l+1}(j)}{\sum_{j=1}^k w_{ij}}, \tag{2}$$

where $w_{ij} = 1/d(P_2^l(i), \tilde{P}_1^l(j))$ and d is a distance metric, i.e., Euclidean distance. Finally, P_2^l can be warped to the first frame using the backward flow:

$$P_w^l = P_2^l + b_{3D}^l. \tag{3}$$

Cost Volume. For the image branch, we follow PWC-Net to construct a partial cost volume by limiting the search range to a maximum displacement of $d_{max} = 4$ pixels around each pixel. The resulting cost volume is organized as a 3D array of dimensions $D^2 \times H \times W$, where H and W are the height and width of the feature map respectively and $D = 2d_{max} + 1 = 9$. For the point branch, we follow PointPWC-Net to construct a learnable cost volume layer. Formally, for a point p, we form the point-to-patch cost by searching its k-NN neighborhoods in the target point cloud. Then the point-to-patch costs are further aggregated with PointConv to construct a patch-to-patch cost volume. The size of the neighborhoods is set to $k = 16$ for all our experiments. Finally, we fuse the cost volumes of the two branches with a Bi-CLFM.

Flow Decoder and Estimator. Our optical flow decoder follows the original PWC-Net which employs a multi-layer CNN with DenseNet [3] connections. The numbers of the feature channels for each convolutional layer are 128, 128, 96, 64, and 32 respectively. Our scene flow estimator follows the original PointPWC-Net, which consists of two PointConv layers with 128 feature channels and a two-layer MLP with 128 and 64 feature channels respectively. Next, the outputs of the optical flow decoder and the scene flow

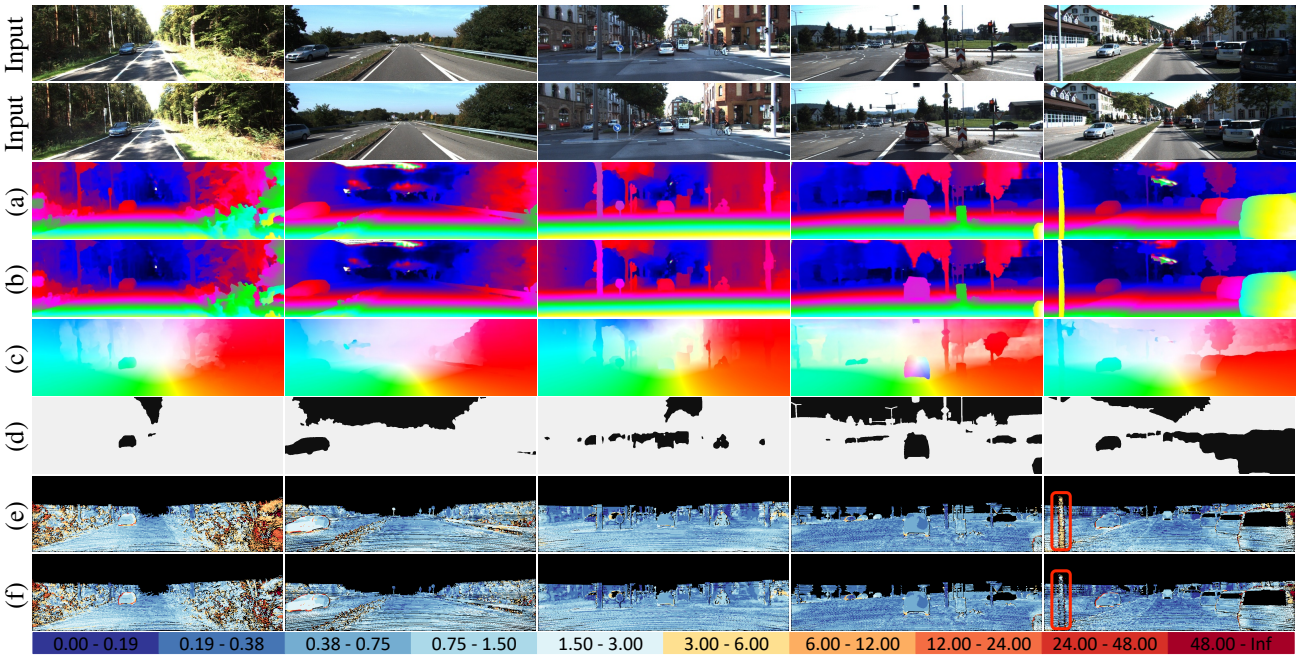


Figure 12. Qualitative results on the validation set of the KITTI Scene Flow dataset: (a) disparity of the first frame by GA-Net [10], (b) warped disparity of the second frame by our point branch, (c) optical flow by our image branch, (d) background segmentation by DDRNet-Slim [2], (e) scene flow error map without leveraging scene rigidity, (f) scene flow error map after the background rigidity refinement.

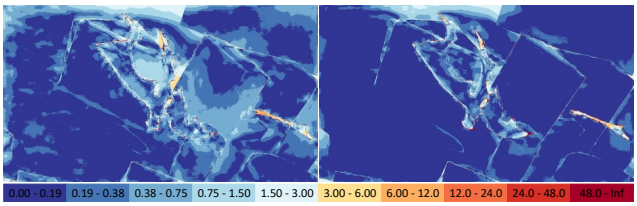
decoder are fused by a Bi-CLFM. Finally, a single-layer perceptron (SLP) with two output channels serves as the optical flow estimator to estimate the optical flow at level l, and the scene flow estimator is an SLP with three output channels to estimate the scene flow at level l. For both the image branch and point branch, weights are shared across all pyramid levels.

2. Experiments

2.1. Training Details

FlyingThings3D. We follow FlowNet3D [5] to lift the disparity maps to point clouds with depth <35m. For data augmentation, we only perform random flipping (for both horizontal and vertical directions) since the number of the training samples is enough. We train our model with the L_2-norm loss function and then fine-tune it on the same dataset with the robust loss function. Fine-tuning with the robust loss fusion gives less penalty to outliers and can improve the threshold metrics (ACC_{1px} and $ACC_{.05}$). In Fig. 13, we provide a visualized comparison of the error map to demonstrate the effect.

KITTI. Using the weight pre-trained on FlyingThings3D, we finetune our model on Driving and KITTI in sequence. Basic data augmentation strategies including color jitter, random horizontal flipping, random scaling, and random cropping are applied. We use the `ColorJitter` from Torchvision [6] with brightness 0.4, contrast 0.4, saturation 0.2, and hue $0.4/\pi$. For Driving, we randomly crop the image with a size of 768x384. For KITTI, we randomly rescale the image by the factor in the range $[1.0, 1.5]$.



(a) W/o Finetune (b) With Finetune

Figure 13. Visualized error map of optical flow on FlyingThings3D with/without finetuning using the robust loss function.

2.2. Additional Qualitative Examples

KITTI. Since the website of KITTI only visualizes a limited number of samples on the test set, we provide additional qualitative examples on the validation split in Fig. 12. Row (a), (b), and (c) are respectively estimated by GA-Net [10], our point branch, and our image branch before the rigidity refinement step. Our model can estimate the foreground motions and most background motions accurately as shown in Row (e). It fails when objects are entirely occluded (see the last column). By applying the rigidity refinement for the background (the segmentation masks are estimated by DDRNet-Slim [2] and are visualized in Row (d)), our model can estimate both foreground and background motions accurately, as shown in Row (f).

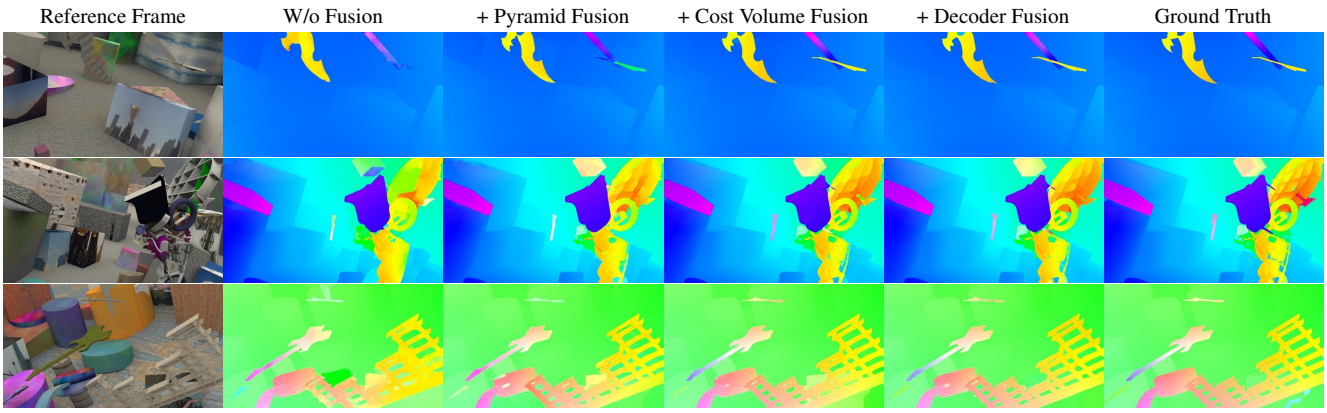


Figure 14. A visual ablation on our multi-stage fusion pipeline. Fusing pyramid features helps to recover the structure and boundary of complex objects. Fusing the cost volume enables the network to capture small and fast-moving objects. Fusing the features of the flow decoder improves the performance on complex scenes where objects overlap.

Ablations for Multi-stage Fusion Pipeline. CamLiFlow performs feature fusion in a multi-stage manner. In the main paper, we confirm the effectiveness of each stage by quantitative results in Tab. 4. Here, we provide a more detailed qualitative analysis. In Fig. 14, the second column denotes a variant of our model where no fusion connection exists between the two branches. The third column only fuses the pyramid feature, while the fourth column further fuses the cost volume. The fifth column denotes our full model which performs feature fusion at all three stages.

As we can see, fusing pyramid features makes the structure of the objects clearer, since point pyramid encodes geometric information which helps to recover the shape of complex objects. Fusing the cost volume enables the model to capture small and fast-moving objects, since the point-based 3D cost volume searches for a dynamic range of neighborhoods and can be complementary to the 2D cost volume which searches for a fixed range. Fusing the features of the flow decoder improves overall performance, especially on complex scenes where objects overlap.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1

[2] Yuanduo Hong, Huihui Pan, Weichao Sun, Yisong Jia, et al. Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. *arXiv preprint arXiv:2101.06085*, 2021. 3

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 2

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 1

[5] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 529–537, 2019. 3

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019. 3

[7] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8934–8943, 2018. 1

[8] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pages 402–419. Springer, 2020. 1

[9] Wenxuan Wu, Zhiyuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. Pointpwc-net: A coarse-to-fine network for supervised and self-supervised scene flow estimation on 3d point clouds. *arXiv preprint arXiv:1911.12408*, 2019. 1

[10] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip HS Torr. Ga-net: Guided aggregation net for end-to-end stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 185–194, 2019. 3