CamLiFlow: Bidirectional Camera-LiDAR Fusion for Joint Optical Flow and Scene Flow Estimation

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

Abstract
In this paper, we study the problem of jointly estimating the optical flow and scene flow from synchronized 2D and 3D data. Previous methods either employ a complex pipeline that splits the joint task into independent stages, or fuse 2D and 3D information in an “early-fusion” or “late-fusion” manner. Such one-size-fits-all approaches suffer from a dilemma of failing to fully utilize the characteristic of each modality or to maximize the inter-modality complementarity. To address the problem, we propose a novel end-to-end framework, called CamLiFlow. It consists of 2D and 3D branches with multiple bidirectional connections between them in specific layers. Different from previous work, we apply a point-based 3D branch to better extract the geometric features and design a symmetric learnable operator to fuse dense image features and sparse point features. Experiments show that CamLiFlow achieves better performance with fewer parameters. Our method ranks 1st on the KITTI Scene Flow benchmark, outperforming the previous art with 1/7 parameters. Code is available at https://github.com/MCG-NJU/CamLiFlow.

1. Introduction
Optical flow and scene flow are the motion field in 2D and 3D space respectively. Through them, we can gain insights into the dynamics of the scene, which are critical to some high-level scene understanding tasks. In this work, we focus on the joint estimation of optical flow and scene flow, which addresses monocular camera frames with sparse depth measurements from LiDAR.

Previous methods [3,31,55,56] construct a modular network that decomposes the estimation of flow into multiple subtasks. These submodules are independent of each other, making it impossible for utilizing their complementarity. Moreover, the limitations of any submodule will hurt the overall performance, since the whole pipeline depends on its results.

* Corresponding author (liuhs@smail.nju.edu.cn).
learnable bidirectional bridge connects the two branches to pass complementary information. Moreover, recent point-based methods [28, 29, 36–38, 48, 51, 52] achieve remarkable progress for 3D computer vision. This inspires us to process point clouds with a point-based branch, which can extract the fine 3D geometric information without any voxelization or projection.

It is worth noting that there are two challenges for the fusion of the image branch and the point branch. First, image features are organized in a dense grid structure, while point clouds do not conform to the regular grid and are sparsely distributed in the continuous domain. As a result, there is no guarantee of one-to-one correspondence between pixels and points. Second, LiDAR point cloud possesses the property of varying density, where nearby regions have much greater density than farther-away regions. To tackle the first problem, we propose a new learnable fusion operator, named bidirectional camera-LiDAR fusion module (Bi-CLFM), which fuses image/point features for both directions via learnable interpolation and sampling. As for the second problem, we propose a new transformation operator, named inverse depth scaling (IDS), which balances the distribution of points by scaling them non-linearly according to the inverse depth.

Experiments demonstrate that our approach achieves better performance with much fewer parameters. On FlyingThings3D [32], we achieve up to a 48.4% reduction in end-point-error over RAFT-3D with only 1/6 parameters. On KITTI [33], CamLiFlow achieves an error of 4.43%, outperforming the previous art [56] with only 1/7 parameters. The leaderboard is shown in Fig. 1.

2. Related Work

**Optical Flow.** Optical flow estimation aims to predict dense 2D motion for each pixel from a pair of frames. Traditional methods [4, 6–8, 16, 50] often formulate optical flow as an energy minimization problem. FlowNet [12] is the first end-to-end trainable CNN for optical flow estimation, which adopts an encoder-decoder architecture. FlowNet2 [20] stacks several FlowNets into a larger one. PWC-Net [43] and some other methods [18, 19, 40, 54] apply iterative refinement using coarse-to-fine pyramids. RAFT [44] constructs 4D cost volumes for all pairs of pixels and updates the flow iteratively. Although achieving state-of-the-art performance, RAFT runs much slower than PWC-Net. Hence, our two branches are built upon the PWC architecture to achieve a better trade-off between accuracy and speed.

**Scene Flow from RGB-D Frames.** RGB-D scene flow is the problem of estimating dense 3D motion for each pixel from a pair of stereo or RGB-D frames. Like optical flow, traditional methods [21, 22, 33, 39] explore variational optimization and discrete optimization and treat scene flow as an energy minimization problem. Recent methods [3, 31, 55, 56] divide scene flow estimation into multiple subtasks and build a modular network with one or more submodules for each subtask. Although achieving remarkable progress, their submodules are independent of each other, which can not exploit the complementary characteristics of different modalities. RAFT-3D [45] concatenate images and depth maps to RGB-D frames at an early stage, followed by a unified 2D network which iteratively updates a dense field of pixel-wise SE3 motion. However, this kind of “early fusion” makes it hard for 2D CNNs to take advantage of the rich 3D structural information.

**Scene Flow from Point Clouds.** Recently, researchers start to study scene flow estimation in 3D point clouds (e.g. from LiDAR) [14, 25, 28, 29, 35, 48, 49, 52]. Based on [38], FlowNet3D [28] uses a flow embedding layer to represent the motion of points. FlowNet3D++ [48] achieves better performance by adding geometric constraints. Inspired by Bilateral Convolutional Layers, HPLFlowNet [14] projects the points onto a permutohedral lattice. PointPWC-Net [52] introduces a learnable cost volume for point clouds and estimates scene flow in a coarse-to-fine fashion. However, these methods do not exploit color features provided by images. As we demonstrate in our experiments, fusing point clouds with images can bring significant improvements.

**Camera-LiDAR Fusion.** Cameras and LiDARs have complementary characteristics, facilitating many computer vision tasks, such as depth estimation [13, 30, 57], scene flow estimation [2, 42], 3D object detection [10, 27, 36, 47, 53], etc. Some researchers [2, 36, 47, 57] build a modular network and perform result-level fusion, while the others [13, 27, 30, 42, 53] explore feature-level fusion schemes including early-fusion and late-fusion. Instead, we propose a multi-stage and bidirectional fusion pipeline, which not only fully utilizes the characteristic of each modality, but maximizes the inter-modality complementarity as well.
3. CamLiFlow

Given a pair of the synchronized camera and LiDAR frames, CamLiFlow jointly estimates dense optical flow for camera frames and sparse scene flow for LiDAR frames. As illustrated in Fig. 3, CamLiFlow consists of two symmetric branches, named image branch and point branch, for 2D and 3D data respectively. Both branches are built on top of the PWC architecture [43, 52] where flow computed at the coarse level is upsampled and warped to a finer level. Features are fused in a bidirectional manner at multiple levels and stages.

In the following sections, we first introduce the bidirectional camera-LiDAR fusion module along with the multi-stage fusion pipeline. Next, we introduce inverse depth scaling, which makes the distribution of points more even across different regions. Finally, a multi-task loss for joint optimization is also introduced.

3.1. Bidirectional Camera-LiDAR Fusion Module

As mentioned above, the fusion between camera and LiDAR is challenging, since the data structures of image features and point features do not match. To overcome this, we introduce bidirectional camera-LiDAR fusion module (Bi-CLFM), which can fuse dense image features and sparse point features in a bidirectional manner.

As illustrated in Fig. 4, Bi-CLFM takes image features $F \in \mathbb{R}^{H \times W \times C_{2D}}$, point features $G = \{g_i|i = 1,\ldots,N\} \in \mathbb{R}^{N \times C_{3D}}$ and point positions $P = \{p_i|i = 1,\ldots,N\} \in \mathbb{R}^{N \times 3}$ as input, where $N$ denotes the number of points. Features are fused for both directions so that both modalities can benefit each other. Note that we stop the gradient at specific locations to prevent one modality from dominating and stabilize the training (please refer to the supplementary material for more details).

2D $\Rightarrow$ 3D. First, points are projected to the image plane (denoted as $X = \{x_i|i = 1,\ldots,N\} \in \mathbb{R}^{N \times 2}$) to retrieve the corresponding 2D feature:

$$H = \{F(x_i)|i = 1,\ldots,N\} \in \mathbb{R}^{N \times C_{2D}},$$

where $F(x)$ denotes the image feature at $x$ and can be retrieved by bilinear interpolation if the coordinate is not an integer. Next, the retrieved feature $H$ is concatenated with the input 3D feature $G$. Finally, a $1 \times 1$ convolution is employed to reduce the dimension of the fused 3D feature.

3D $\Rightarrow$ 2D. Similarly, points are first projected to the image plane (denoted as $X = \{x_i|i = 1,\ldots,N\} \in \mathbb{R}^{N \times 2}$). Since point clouds are sparse, we propose fusion-aware interpolation (detailed in the following paragraphs) to create a dense feature map $D \in \mathbb{R}^{H \times W \times C_{3D}}$ from sparse 3D feat-
We empirically test it in the ablation study (Fig. 9). The fusion-aware interpolation module makes it more robust in complex scenes with overlapping objects, since dense 2D features can be used to guide the densification of the sparse 3D features. Thus, features are fused by a Bi-CLFM at multiple levels for complementarity.

**Warping.** At each pyramid level $l$, both image features and point clouds are warped towards the reference frame using the upsampled flow from the lower level. Since the warping layer does not introduce any learnable parameters, we do not perform feature fusion after this stage.

**Cost Volume.** Cost volume stores the matching costs between the reference frame and the warped target frame. For the image branch, we follow [43] to construct a partial cost volume by limiting the search range to 4 pixels around each pixel. For the point branch, we follow [52] to construct a learnable cost volume layer. The pixel-based 2D cost volume maintains a fixed range of neighborhoods, while the point-based 3D cost volume searches for a dynamic range. Hence, we fuse the two cost volumes with a Bi-CLFM.

**Flow Estimator.** We build a flow estimator for each branch. The input of the flow estimator includes the cost volume, the features of the reference frame, and the upsampled flow. Our optical flow estimator follows [43], which employs a multi-layer CNN with DenseNet [17] connections. Our scene flow estimator follows [52], which is built as multiple layers of PointConv [51]. Features from the second last layer of the two estimators are fused. For clarity, we refer to the last layer as the “flow estimator” and the other layers as the “flow decoder” in Fig. 3.

**3.3. Inverse Depth Scaling**

As mentioned above, the distribution of LiDAR point clouds is not balanced, where nearby region has much greater density than farther-away region. Here, we propose a transformation operator for point clouds to address the problem, named inverse depth scaling (IDS). Formally, let $(P_x, P_y, P_z)$ and $(P'_x, P'_y, P'_z)$ be the coordinate of a point before and after the transformation respectively. IDS scales all three dimensions equally by the inverse depth $\frac{1}{P_z}$:

$$\frac{\delta P'_x}{\delta P_x} = \frac{\delta P'_y}{\delta P_y} = \frac{\delta P'_z}{\delta P_z} = \frac{1}{P_z}. \quad (4)$$

The transformed coordinates $(P'_x, P'_y, P'_z)$ can be inferred by integrating the above formula:

$$P'_x = \int \frac{1}{P_z} dP_x = P_z + C_x, \quad (5)$$

$$P'_y = \int \frac{1}{P_z} dP_y = P_y + C_y, \quad (6)$$

$$P'_z = \int \frac{1}{P_z} dP_z = \log P_z + C_z, \quad (7)$$

Figure 5. Details of Fusion-Aware Interpolation. For each target pixel, we find the $k$ nearest points around it. A learnable MLP followed by MEAN is employed to aggregate features. The “interpolated” point features are concatenated with the input image features, followed by a $1 \times 1$ convolution to reduce the feature dimension.

**Fusion-Aware Interpolation.** To solve the problem of fusing sparse point features into dense image features, we propose a learnable fusion-aware interpolation. As illustrated in Fig. 5, for each target pixel $q$ in the dense map, we find its $k$ nearest neighbors among the projected points over the image plane. An MLP followed by MEAN is used to aggregate features, which can be formulated as:

$$D(q) = \frac{1}{k} \sum_{x_i \in N_q} \text{MLP}([x_i - q, S(q, x_i), g_i]), \quad (2)$$

where $N_q$ denotes all the neighborhood points, $g_i$ is the 3D feature of point $i$ and $\cdot$ denotes concatenation. The inputs of our MLP also include 2D similarity measurements between the $q$ and its neighbors, which is defined as:

$$S(q, x_i) = F(q) \cdot F(x_i). \quad (3)$$

Introducing 2D similarity measurements into the interpolation module makes it more robust in complex scenes with overlapping objects, since dense 2D features can be used to guide the densification of the sparse 3D features. We empirically test it in the ablation study (Fig. 9).

**3.2. Multi-stage Fusion Pipeline**

In this section, we build a multi-stage and bidirectional fusion pipeline with Bi-CLFM. Our backbone is based on the PWC architecture, which consists of multiple stages including feature extraction, warping, cost volume, and flow estimation. Within each stage, the two modalities are learned in separate branches using modality-specific architecture. At the end of each stage, a Bi-CLFM connects the two branches to pass complementary information.

**Feature Pyramid.** Given a pair of images and point clouds, we generate a feature pyramid for the image branch and the point branch respectively (the configuration details are included in the supplementary material). For each level $l$, image features are downsampled by a factor of 2 using residual blocks, while points are downsampled by the same factor using furthest point sampling, followed by a PointConv [51] to aggregate features. The image pyramid encodes textural information, while the point pyramid encodes geometric information. Thus, features are fused by a Bi-CLFM at multiple levels for complementarity.
where both $C_x$ and $C_y$ are set to 0, and $C_z$ is set to 1 to avoid zero depth.

In Fig. 6, we perform a statistic on FlyingThings3D and the raw Velodyne data of KITTI to show the density of points across different distances with/without IDS. The local density around a point is measured by averaging the offsets of its $k$ nearest neighbors. As we can see, IDS makes the distribution of points more even across different regions. In this paper, point clouds are transformed by IDS before being sent to the neural network.

### 3.4. Multi-task Loss

Although the estimation of optical flow and scene flow are highly relevant (the projection of scene flow onto the image plane becomes optical flow), we formulate them as two different tasks. We supervise the 2D and 3D branches respectively and design a multi-task loss for joint optimization. Let $\hat{f}_{2D}^q$ and $\hat{f}_{3D}^q$ be the ground truth optical flow and scene flow at the $q$th level respectively. The regression loss for each branch is defined as follows:

$$L_{2D} = \sum_{l=t_0}^{L} \alpha_l \sum_{x} \| f_{2D}^l(x) - \hat{f}_{2D}^l(x) \|_2,$$  \hspace{1cm} (8)

$$L_{3D} = \sum_{l=t_0}^{L} \alpha_l \sum_{p} \| f_{3D}^l(p) - \hat{f}_{3D}^l(p) \|_2,$$  \hspace{1cm} (9)

where $\| \cdot \|_2$ computes the $L_2$ norm. For fine-tuning, we use the following robust training loss:

$$L_{2D} = \sum_{l=t_0}^{L} \alpha_l \sum_{x} (| f_{2D}^l(x) - \hat{f}_{2D}^l(x) | + \epsilon)^q,$$ \hspace{1cm} (10)

$$L_{3D} = \sum_{l=t_0}^{L} \alpha_l \sum_{x} (| f_{3D}^l(x) - \hat{f}_{3D}^l(x) | + \epsilon)^q,$$ \hspace{1cm} (11)

where $| \cdot |$ computes the $L_1$ norm, $q = 0.4$ gives less penalty to outliers and $\epsilon$ is set to 0.01. The final loss is a weighted sum of the losses defined above:

$$L = L_{2D} + \lambda L_{3D},$$ \hspace{1cm} (12)

where $\lambda$ is set to 1.0 for all our experiments.

### 4. Experiments

We implement our model using PyTorch [34]. For all experiments we use the Adam optimizer [24] with weight decay set to $10^{-6}$. The loss weights are set to $\alpha_0 = 8, \alpha_1 = 4, \alpha_2 = 2, \alpha_3 = 1$, and $\alpha_4 = 0.5$.

#### 4.1. Main Results

We evaluate our method on the synthetic dataset FlyingThings3D [32] and the real-world dataset KITTI [33]. FlyingThings3D consists of stereo and RGB-D images rendered with multiple randomly moving objects from ShapeNet [9], which is large-scale and challenging. KITTI Scene Flow is a real-world benchmark for autonomous driving, consisting of 200 training scenes and 200 test scenes.

##### 4.1.1 FlyingThings3D

**Data Preprocessing.** Following previous work [14, 20, 52], we use the subset of FlyingThings3D. The training and validation set respectively contains 19640 and 3824 pairs of camera-LiDAR frames. We follow FlowNet3D [28] instead of HPLFlowNet [14] to lift the depth images to point clouds, since HPLFlowNet only keeps non-occluded points which oversimplifies the problem.

**Training.** The training consists of two stages. First, we train our model for 600 epochs with the $L_2$-norm loss function. The initial learning rate is set to $4 \times 10^{-4}$ and reduced by half at 400 and 500 epochs. Next, we fine-tune our model for another 800 epochs with the robust loss function and a fixed learning rate of $10^{-4}$. The batch size is set to 32.

**Evaluation Metrics.** Following RAFT-3D, we evaluate our network using 2D and 3D end-point error (EPE), as well as threshold metrics (ACC1px and ACC.05), which measure the portion of error within a threshold.

**Quantitative Results.** In Tab. 1, we compare to several state-of-the-art methods which utilize different input modalities. By fusing the two modalities of camera and LiDAR, our method outperforms all image-only and LiDAR-only methods by a large margin. Our method also outperforms RAFT-3D, which has 45M parameters and takes dense RGB-D frames as input. In contrast, our model is much more lightweight with 7.7M parameters and only requires sparse depth measurements. Moreover, our model reduces the best published EPE3D from 0.062 to 0.032, which proves the superior performance of the point branch.

**Qualitative Results.** The visual comparison of optical flow and scene flow estimation is shown in Fig. 7. We also add two single-modal variations of our method for comparison, which removes the 2D branch or the 3D branch. As we can see, our full model better handles objects with repetitive structures and complex scenes with overlapping objects.
Figure 7. Visualized optical flow and scene flow estimation on the “val” split of the FlyingThings3D subset. The outliers are marked as red for scene flow estimation. Our full model better handles objects with repetitive structures and texture-less regions.

Table 1. Performance comparison on the “val” split of the FlyingThings3D subset. For 2D metrics, we evaluate on full images excluding extremely fast moving regions with flow > 250px. For 3D metrics, we follow the setup of FlowNet3D in which only non-occluded points with depth < 35m are considered for evaluation.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>2D Metrics EPE2D</th>
<th>3D Metrics EPE3D</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlowNet2.0 [20]</td>
<td>Image</td>
<td>5.05</td>
<td>-</td>
<td>162.5M</td>
</tr>
<tr>
<td>PWC-Net [43]</td>
<td>Image</td>
<td>6.55</td>
<td>-</td>
<td>9.4M</td>
</tr>
<tr>
<td>RAFT [44]</td>
<td>Image</td>
<td>3.12</td>
<td>-</td>
<td>5.3M</td>
</tr>
<tr>
<td>FlowNet3D [28]</td>
<td>LiDAR</td>
<td>-</td>
<td>0.151</td>
<td>1.2M</td>
</tr>
<tr>
<td>PointPWC [52]</td>
<td>LiDAR</td>
<td>-</td>
<td>0.112</td>
<td>5.3M</td>
</tr>
<tr>
<td>FLOT [35]</td>
<td>LiDAR</td>
<td>-</td>
<td>0.170</td>
<td>0.1M</td>
</tr>
<tr>
<td>DeepLiDARFlow [42]</td>
<td>Image+LiDAR</td>
<td>6.04</td>
<td>-</td>
<td>8.3M</td>
</tr>
<tr>
<td>RAFT-3D [45]</td>
<td>Image+Depth</td>
<td>2.37</td>
<td>0.062</td>
<td>45M</td>
</tr>
<tr>
<td>Ours (W/o fine-tuning)</td>
<td>Image+LiDAR</td>
<td>2.18</td>
<td>0.033</td>
<td>7.7M</td>
</tr>
<tr>
<td>Ours</td>
<td>Image+LiDAR</td>
<td>2.20</td>
<td><strong>0.032</strong></td>
<td><strong>9.26%</strong></td>
</tr>
</tbody>
</table>

4.1.2 KITTI

**Training.** Using the weight pre-trained on FlyingThings3D and Driving [32], we fine-tune our model on KITTI for 300 epochs with a fixed learning rate of $5 \times 10^{-5}$ and a batch size of 8. We follow [55, 56] and divide the 200 training images into train, val splits based on the 4:1 ratio. During training, we lift the ground-truth disparity maps into point clouds using the provided calibration parameters. Basic data augmentation strategies including color jitter, random horizontal flipping, and random cropping are applied.

**Testing.** During testing, since neither disparity maps nor point clouds are provided, we employ GA-Net [58] to estimate the disparity from stereo images, and generate point clouds with depth < 90m. The sparse output of our point branch is interpolated to create a dense prediction.

**Refinement of Background Scene Flow.** Since most background objects in KITTI are rigid (e.g. ground, buildings, etc), we can refine the background scene flow using a rigidity refinement step. Specifically, we employ DDRNet-Slim [15], a light-weight 2D semantic segmentation network, to determine the rigid background. DDRNet-Slim is pre-trained on Cityscapes [11] and fine-tuned on KITTI. Next, we estimate ego-motion by fitting and decomposing essential matrices from the background flow map using a neural-guided RANSAC [5]. Finally, the background scene flow is refined using the ego-motion and the disparity of the first frame.

**Comparison with State-of-the-art Methods.** We submit our approach to the website of KITTI Scene Flow benchmark and report the leaderboard in Tab. 2. A visualized comparison is shown in Fig. 8. Our approach outperforms all published methods, including RigidMask [56] (SF-all: 4.43% vs. 4.89%), which employs more than 140M parameters. In contrast, our method is much more lightweight with only 19.7M parameters (6.3M GA-Net + 7.7M Cam-
LiFlow + 5.7M DDRNet-Slim). Moreover, previous methods leverage more strict rigid-body assumptions by assigning rigid motions to all objects, while our method can handle general non-rigid motions since we only apply rigid motion refinement to the static background.

If the rigidity refinement step of the background scene flow is removed (corresponding to our “non-rigid” variation in Tab. 2), our method still ranks second on the leaderboard (SF-all: 5.62%). In this setting, our method does not require the background segmentation labels and can deal with any non-rigid motion (no matter foreground or background). Instead, RigidMask fails to handle non-rigid motions and suffers from the limitations of the motion segmentation network, since the whole pipeline depends on its results.

4.2. Ablation Study

In this section, we conduct ablation studies on FlyingThings3D to confirm the effectiveness of each module. All variations are trained for the first stage without fine-tuning with the robust loss function.

**Unidirectional Fusion vs. Bidirectional Fusion.** Cam-LiFlow fuses features in a bidirectional manner. Here, we train two variations where features are fused in a unidirectional manner (2D ⇒ 3D or 2D ⇐ 3D). As shown in Tab. 3, unidirectional fusion either improves 2D metrics or 3D metrics, while bidirectional fusion provides better results for both modalities. Moreover, compared with unidirectional fusion, bidirectional fusion improves the best EPE2D from 2.25 to 2.18 and EPE3D from 0.036 to 0.033, suggesting that the improvement of one modality can also benefit the other.

**Early/Late-Fusion vs. Multi-Stage Fusion.** As mentioned above, flow estimation typically consists of several stages including feature extraction, cost volume, and feature decoding. Here, we verify the effectiveness of feature fusion for each stage, as shown in Tab. 4. The top row denotes the version where no fusion connection exists between the two branches. Both “early-fusion” and “late-fusion” (row 2, 3, 4) can only provide sub-optimal results. In contrast, fusing features at all three stages brings significant improvements compared to “early/late-fusion” (see the supplementary material for more details).
In this paper, we introduce CamLiFlow, a deep neural network for joint optical flow and scene flow estimation. It consists of 2D and 3D branches with multiple bidirectional connections between them in specific layers. Experiments show that CamLiFlow outperforms the previous art with fewer parameters.

Acknowledgements. This work is supported by National Natural Science Foundation of China (No.62076119, No.61921006), Program for Innovative Talents, Entrepreneur in Jiangsu Province, and Collaborative Innovation Center of Novel Software Technology and Industrialization.
References


