FlowStep3D: Model Unrolling for Self-Supervised Scene Flow Estimation

Yair Kittenplon[1] Yonina C. Eldar[2] Dan Raviv[1]
[1]Tel Aviv University [2]Weizmann Institute of Science
{yairk@mai1, darav@tauex}.tau.ac.il yonina.eldar@weizmann.ac.il

Abstract

Estimating the 3D motion of points in a scene, known as scene flow, is a core problem in computer vision. Traditional learning-based methods designed to learn end-to-end 3D flow often suffer from poor generalization. Here we present a recurrent architecture that learns a single step of an unrolled iterative alignment procedure for refining scene flow predictions. Inspired by classical algorithms, we demonstrate iterative convergence toward the solution using strong regularization. The proposed method can handle sizeable temporal deformations and suggests a slimmer architecture than competitive all-to-all correlation approaches. Trained on FlyingThings3D synthetic data only, our network successfully generalizes to real scans, outperforming all existing methods by a large margin on the KITTI self-supervised benchmark.[1]

1. Introduction

Understanding motion is fundamental to many applications in a variety of fields, such as human-computer interaction, robotics, and autonomous driving. The information absorbed within a temporal window is not only a collection of images or a representation of an outcome, but also a description of a process.

Decades ago, computer vision tackled the task of motion estimation, searching for a flow between two images [3, 7, 14, 22, 41]. One significant leap forward in understanding the motion of a scene, defined as scene flow, is the presence of 3D geometry. It liberates us from considering color as the main correspondence feature and allows examining the structure itself to understand the motion. Axiomatic concepts of rigidity [2, 6] provided fast and accurate results, but once piece-wise movements [9, 10, 39] or non-rigidity [1, 15, 23] was allowed, scene flow estimation problem became ill-posed and unfortunately hard to solve.

The rise of artificial intelligence [19] gives hope that solving the 3D flow estimation problem is possible using a network architecture. Indeed, in the last few years, an improvement in different learning-based methods [12, 21, 31, 37, 43] has been seen, outperforming those that relied on optimization. More importantly, these learned models are fast and robust.

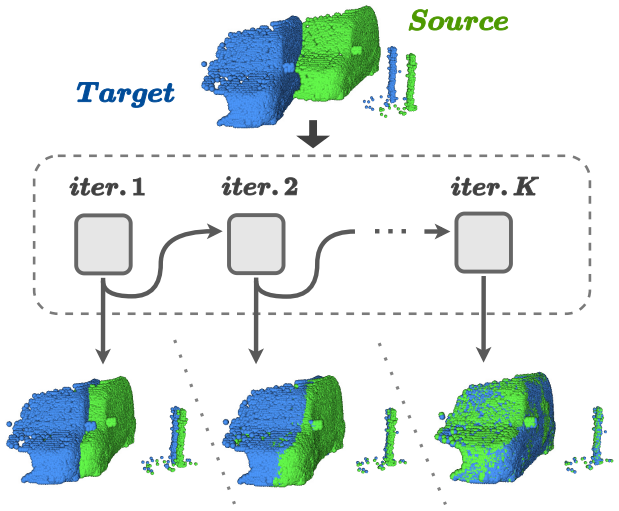

Figure 1. **Model unrolling.** Top: source (green) and target (blue) input point clouds sampled from the KITTI [27] scene-flow dataset. Middle: illustration of our FlowStep3D architecture unrolled for K iterations. Bottom: source warped over the predicted flow at each iteration toward the static target at inference time.

Scene flow estimation is an integral component in the autonomous driving industry, where LiDAR data is used for the perception of the environment. However, LiDAR sensors suffer from sparseness, directly affecting deep-learning flow algorithms that require knowledge of the objects' spatio-temporal neighborhood. In other words, once the structures do not heavily overlap, the process fails. In an attempt to solve this limitation, we recently have seen all-to-all mechanisms both for images [36] and geometry [31]. However, these methods consume large amounts of memory and tend to produce outliers, as now nearby points can be aligned with inconsistent temporal positions.

In this work, we focus on the scene flow problem, where

[1] https://github.com/yairkit/flowstep3d

large deviations between the scenes can occur. A small set of points is used to guide the alignment in an all-to-all approach, and a recurrent refinement block is then unrolled to learn movement differentiators. We train our network to predict a single step at a time and converge iteratively toward the end flow solution, as illustrated in Fig.1. Although unrolled for K iterations during training, our network can be used for inference with a larger number of iterations to handle more significant and complicated deformations.

Trained on synthetic data only, our method improves the state-of-the-art results on the self-supervised KITTI benchmark by a considerable margin. Our architecture is further tested in a fully-supervised framework and achieves slightly better results compared to prior art while benefiting from memory efficiency.

The key contributions of this work are as follows:

- We present the first recurrent architecture for non-rigid scene flow.
- We provide a slim memory all-to-all correlation pipeline by merging low-resolution correlation with an unrolling iterative refinement process.
- Our proposed network achieves large improvements over existing self-supervised methods on both FlyingThings3D and KITTI benchmarks.

2. Related Work

Scene Flow Estimation on Point Clouds. Scene flow estimation was first introduced in [38], who suggested to compute a 3D scene flow from 2D optical flow using a linear algorithm. Later approaches used stereo sequences [16], RGB-D [13], and LiDAR [6]. With the rise of new methods for deep learning on point clouds [32, 33, 35, 42] and the increasing popularity of range data at the autonomous driving domain, more recent approaches suggest learning the 3D scene flow directly from the raw data spatial positions.

Liu *et al*. [21] were the first to introduce a correlation layer that aggregates features of different point clouds based on PointNet[32]. However, the correlation layer was applied at a particular scale, only capturing the correlation of that specific level of features between the point clouds and a fixed neighborhood radius, allowing a small deformation between the point clouds. Gu *et al*. [12] tackled those limitations by introducing multi-resolution correlation layers and suggested using a Bilateral Convolutional layer [17, 18, 35]. Inspired by classical pyramid approaches, Wu *et al*. [43] further improved multi resolution flows by applying it in a coarse to fine manner and showed superior results. However, multi-resolution methods require many learnable parameters and are limited to deformations smaller than the correlation neighborhood. In [37], the authors suggested splitting the movement into rigid ego-motion and non-rigid

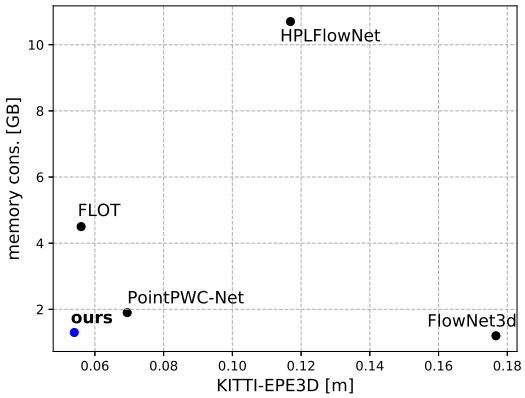

Figure 2. **Memory consumption and accuracy trade-off.** Average end-point-error (Sec.6) of the leading fully-supervised methods on the KITTI test vs. memory consumption on inference with 8192 points per scene.

refinement components, relying on the same architecture as [12]. A different approach focusing on all-to-all correlation suggested by [31], using optimal transport tools to estimate the scene flow, showed excellent results. However, an all-to-all correlation matrix for a large scale point cloud is inefficient.

We adopt the all-to-all correlation concept, but unlike [31], we suggest to use it efficiently in a much deeper, lower resolution space.

Self-supervised Learning. Learning to estimate scene flow in a self-supervised manner is an active field of research. Mittal *et al*. [29] showed that cycle-consistency and nearest-neighbor losses could be used for self-supervision of scene flow learning, using a backbone of FlowNet3D [21], pre-trained in a fully supervised manner on synthetic data. Tishchenko *et al*. [37] combined the same self-supervised losses with a fully supervised loss into a 'Hybrid loss', while [43] suggested a fully self-supervised process, combining Chamfer, nearest-neighbors, and laplacian losses.

We follow [43] and choose the Chamfer loss as the data-term loss of our training. Still, inspired by classical non-rigid alignment algorithms, we claim that this data term is not sufficient for a one-shot, end-to-end solution. Hence we suggest an iterative approach for the scene flow estimation and emphasize the need for strong regularization loss term.

Algorithm Unrolling. While the vast majority of deep learning approaches propose a purely data-driven, one-shot solution, there is a rising trend of combining iterative algorithms to neural network architectures to take advantage of both learning and prior knowledge. Recent works showed promising results for signal and image processing tasks [8, 20, 25, 28, 30, 44, 45] by unrolling either an explicit

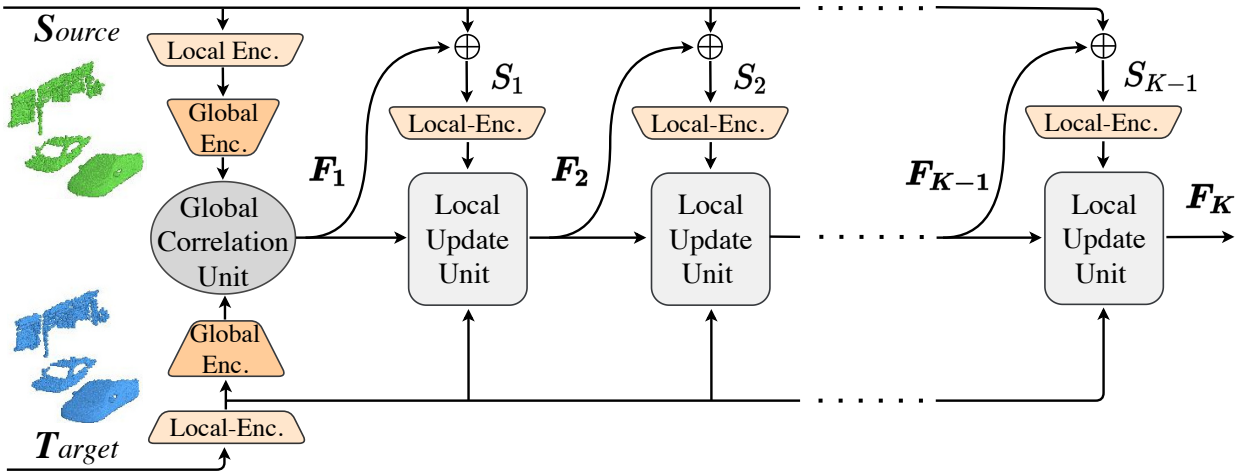

Figure 3. **FlowStep3D high-level overview.** At the first iteration, the global correlation unit (Fig. 4) produces the initial flow F_1 based on source's and target's global features obtained by deep encoding. In each iteration, the source point cloud is warped toward the target by adding the predicted flow from the previous iteration F_{k-1}. It is then locally encoded and fed into a local update unit (Fig. 5) to refine the flow estimation. The weights of the local update unit and of the encoders are shared across all their appearances.

iterative solution for an energy minimization problem or a model. A contemporary approach named RAFT [36] suggested model unrolling for 2D optical flow estimation, performing lookups on a 4D all-to-all correlation volume.

We suggest to unroll a single-step flow estimation model. Inspired by [36], we also adopt the idea of using a gated recurrent unit for iterative updates. An essential concept of our method, which is different from [36], is the computation of new features for the warped scene at every iteration. It is necessary since all point cloud convolution methods are not rotation invariant, so the features of the source change as it is being rotated toward the target. We consider this process as a critical component to learning differentiators iteratively.

3. Problem Definition

Scene flow is the 3D motion field of points in a scene. For a given two sets of points $S = \{p_i \in \mathbb{R}^3\}_{i=1}^{n_1}$ and $T = \{q_j \in \mathbb{R}^3\}_{j=1}^{n_2}$, sampled from a dynamic scene at two consecutive time frames, we denote by $f_i \in \mathbb{R}^3$ the translational motion vector of a point $p_i \in S$ from the first frame toward its new location in the second frame. Our goal is to estimate the scene flow $F = \{f_i\}_{i=1}^{n_1}$ that describes the best non-rigid transformation, which aligns S toward T. Due to both the sparsity of the 3D data and possible occlusions, a point p_i' may not be presented in T. Therefore, we do not learn the correspondence between S and T, but a flow representation for each point $p_i \in S$.

In general, every point p_i, q_j, may have additional information such as color or geometric features. The number of points in the source may differ from the number of points in the target, i.e., n_1 and n_2 are not necessarily equal.

4. Architecture

We suggest an iterative system (Fig. 3) that predicts a flow sequence $\{\boldsymbol{F}_1, ..., \boldsymbol{F}_K\}$, where $\boldsymbol{F}_K = \boldsymbol{F}^*$ is our final flow estimation. First, we use a global correlation unit (Sec. 4.2) to guide the alignment in an all-to-all approach. Next, we unroll a local update unit (Sec. 4.3), to learn movement refinements. Our local update unit implements a single conceptual iteration of an Iterative-Closest-Point (ICP) algorithm [4, 2], replacing the two phases (a. finding correspondence and b. estimating the best smooth transformation based on that correspondence) by learned components.

The number of iterations K is a hyper-parameter and can be larger during inference than during training to handle more complicated and large deformations, as discussed in Sec. 6.3.

4.1. Local And Global Features Encoding

Local features of a point encode the geometric features of its relatively small neighborhood and are useful for local alignment refinements. On the other hand, global features capture high-level information regarding the relative position of the point in the scene, using a larger receptive field and deeper encoding. A crucial part of our method is the distinction between the local and the global features of a point cloud.

We use the *set_conv* layer suggested by FlowNet3D [21] as our convolution mechanism and *furthest_point_sampling* method for down-sampling. Our local encoder $g_\theta : \mathbb{R}^{n\times 3} \mapsto \mathbb{R}^{n'\times d_{local}}$ consists of only two *set_conv* layers, capturing a relatively small receptive field, so that its output encodes an input point clouds shallow features of dimension d_{local}, at resolution n'. Local encoding is first applied on

both source and target input point clouds to produce $g_\theta(S)$, $g_\theta(T)$, and then applied again at every iteration k on the warped source point cloud S_k producing $g_\theta(S_k)$.

In order to extract global features, the local features descriptors $g_\theta(S)$, $g_\theta(T)$ are injected into an additional encoder $h_\theta : \mathbb{R}^{n' \times d_{local}} \mapsto \mathbb{R}^{n'' \times d_{global}}$, which produces $h_\theta(g_\theta(S))$ and $h_\theta(g_\theta(T))$, a deeper representation of S and T of dimension d_{global} and resolution $n'' << n'$, which we denote as $\tilde{h}_\theta(S)$, $\tilde{h}_\theta(T)$ to ease notations. Both g_θ and h_θ encoders have shared weights across all their appearances.

4.2. Global Correlation Unit

We use a global correlation unit to estimate the initial scene flow $\boldsymbol{F_1}$ based on a deep, coarse all-to-all mechanism, illustrated in Fig. 4.

Coarse All-to-all Correlation Matrix. As the first step of our global correlation unit, we use $\tilde{h}_\theta(S)$, $\tilde{h}_\theta(T)$ to calculate a coarse all-to-all correlation matrix $M \in \mathbb{R}^{n_1'' \times n_2''}$. Inspired by FLOT [31], we calculate cosine similarity between the features vectors:

$$sim(i,j) = \frac{\tilde{h}_\theta(p)_i^T \tilde{h}_\theta(q)_j}{\|\tilde{h}_\theta(p)_i\|_2 \|\tilde{h}_\theta(q)_j\|_2}, \tag{1}$$

and then use an exponential function to derive from it a soft correlation matrix:

$$M_{i,j} = \exp\left(\frac{sim(i,j) - 1}{\epsilon}\right). \tag{2}$$

Thus, every entry $M_{i,j}$, describes the correlation between $\tilde{h}_\theta(p)_i$ and $\tilde{h}_\theta(q)_j$, and the softmax temperature ϵ is a hyper-parameter, set to 0.03.

Unlike [31], we calculate the all-to-all correlation matrix M at a lower dimension, $n_1'' \times n_2'' << n_1 \times n_2$, thereby significantly reduce the required memory.

Global Flow Estimation. In order to use the calculated correlation matrix for global flow embedding in the euclidean space, we apply a simple matrix multiplication:

$$\tilde{f}_i = \frac{\sum_{j=1}^{n_2''} M_{i,j} \tilde{q}_j}{\sum_{j=1}^{n_2''} M_{i,j}} - \tilde{p}_i. \tag{3}$$

Thus, $\tilde{f}_i$ is the average distance between $\tilde{p}_i \in \tilde{S}$ to all points $\{\tilde{q}_j\}_{j=1}^{n_2''} \in \tilde{T}$, weighted by their correlation magnitudes, where $\tilde{S} \in \mathbb{R}^{n_1'' \times 3}$ and $\tilde{T} \in \mathbb{R}^{n_2'' \times 3}$ are the coarse versions of S and T coordinates after the the encoder's downsampling. The first iteration flow $\boldsymbol{F_1} \in \mathbb{R}^{n_1 \times 3}$, is regressed out of $\tilde{\boldsymbol{F}}_\mathbf{1} = \{\tilde{f}_i\}_{i=1}^{n_1''}$ by a set of *set_up_conv* layers [33].

4.3. Local Update Unit

We use an iterative update procedure, starting at the global flow estimation $\boldsymbol{F_1}$, and estimating the rest of the flow sequence $\{\boldsymbol{F}_2, ..., \boldsymbol{F}_K\}$ based on local information.

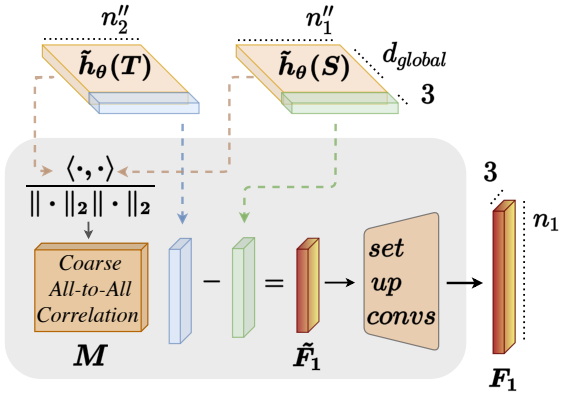

Figure 4. **Global Correlation Unit.** The inner-product of $\tilde{h}_\theta(S)$ and $\tilde{h}_\theta(T)$, the global features of the source and target, is used to create coarse all-to-all correlation matrix M. Matrix multiplication followed by a set of *set_up_conv* layers are then used to predict the global flow F_1, as described in Sec. 4.2.

Warp and Encode. At each iteration $k \in \{2, .., K\}$, we use the estimated flow from the previous iteration for warping the points of the source, i.e $S_{k-1} = S + F_{k-1}$. Next, using the local encoder g_θ, we extract a new local features descriptor for the warped source $g_\theta(S_{k-1})$, which we will later use for local correlation calculation (Fig. 3 top).

Local Correlation. To derive the correlation between the local features of the warped source and the target, we adopt the *flow_embedding* correlation layer proposed by FlowNet3D [21]. The proposed correlation layer aggregates feature similarity and spatial relationships of points within a local neighborhood, and therefore is suitable for local refinements. Specifically, at each iteration k, we calculate *flow_embedding*$(g_\theta(S_{k-1}), g_\theta(T)) \in \mathbb{R}^{n_1' \times d_{corr}}$, which encodes flow embedding for every point in the warped source S_{k-1} toward the target T.

Gated Recurrent Unit (GRU). Inspired by RAFT [36], we use a gated activation unit based on the design of a GRU cell [5] as our updating mechanism. Given previous iteration hidden state h_{k-1}, together with current iteration information x_k, it produces an updated hidden state h_k:

$$z_k = \sigma(set\_conv_z([h_{k-1}, x_k])), \tag{4}$$

$$r_k = \sigma(set\_conv_r([h_{k-1}, x_k])), \tag{5}$$

$$\tilde{h}_k = tanh(set\_conv_h([r_k \odot h_{k-1}, x_k])), \tag{6}$$

$$h_k = (1 - z_k) \odot h_{k-1} + z_k \odot \tilde{h}_k, \tag{7}$$

where $\odot$ is the Hadamard product, $[\cdot, \cdot]$ is a concatenation and $\sigma(\cdot)$ is the sigmoid activation function.

We define $x_k \in \mathbb{R}^{n_1' \times (d_{local} + d_{corr} + 3 + d_{motion})}$ to be the concatenation of the warped source's local features, local flow embedding, previous iteration flow, and previous iteration flow's features. The previous iteration flow's features

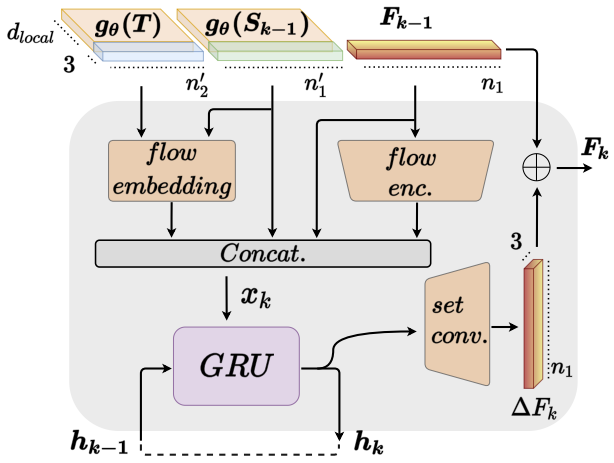

Figure 5. **Local Update Unit.** The previous iteration predicted flow F_{k-1} and its encoding features, together with local features descriptors of the current state source $g_\theta(S_{k-1})$, and of the target $g_\theta(T)$, and their correlation (*flow_embedding*$(\cdot,\cdot)$), are concatenated into x_k. Gated Recurrent Unit (GRU) followed by *set_up_conv* layers produces flow refinement ΔF_k, as described in Sec. 4.3.

are obtained by passing it through two *set_conv* layers called *flow_enc*: $\mathbb{R}^{n_1\times 3} \mapsto \mathbb{R}^{n'_1\times d_{motion}}$, as shown in Fig. 5.

For the initialization of the first iteration's hidden state, we pass the local features of the source point cloud $g_\theta(S)$, through two *set_conv* layers.

Scene flow prediction. Given the new hidden state h_k produced by the GRU cell, we use a flow regressor consisting of two *set_conv* layers to estimate the flow refinement ΔF_k. The updated flow is then calculated as $\boldsymbol{F_k} = F_{k-1} + \Delta F_k$.

Regression of flow refinements at totally different scales by the same CNN component is challenging. Hence, to encourage our system to learn coarse displacements in first iterations, we multiply the magnitude of each predicted scene flow by a factor $\frac{1}{(C\cdot(k-1)+1)}$, where C is a hyper-parameter.

5. Training Loss Functions

To train our iterative system, we unroll K iterations and apply a loss function for each iteration prediction $\boldsymbol{F}_k$:

$$\mathcal{L}_{seq.} = \sum_{k=1}^{K} \mathcal{L}_k. \tag{8}$$

Each iteration loss $\mathcal{L}_k$ in the sequence can be chosen to be a self-supervised (Sec. 5.1) or a fully-supervised (Sec. 5.2) loss.

5.1. Self-supervised Loss

Due to the lack of labeled data for 3D scene flow, we designed our solution to be trained in a self-supervised manner, i.e. without the need of ground-truth flow.

Chamfer Loss. We follow previous works [11, 40, 43] and choose the Chamfer distance, which enforces the source to move toward the target according to mutual closest points, as our self-supervised data loss:

$$\mathcal{L}_k^{ch.} = \mathcal{D}_{ch.}(S_k, T) \overset{\text{def}}{=} \sum_{p\in S_k} \min_{q\in T} \|p-q\|_2^2 + \sum_{q\in T} \min_{p\in S_k} \|q-p\|_2^2 \tag{9}$$

where $S_k := S + F_k$ is the warped source according to the predicted flow at iteration k.

Regularization Loss. Since Chamfer distance has multiple local minima, it is crucial to regularize it in order to reach sufficient convergence. Another reason for which our system requires strong regularization is that we warp the source according to the predicted flow before encoding it again. Hence, we need to carefully preserve the objects' structures so that encoding the warped scene will produce meaningful local geometric features (Fig.8).

Motivated by [1, 11, 34, 43], we propose a strong Laplacian regularization, i.e we enforce the source to preserve its Laplacian when warped according to the predicted flow:

$$\begin{gathered} \mathscr{L}(S+F_k) \simeq \mathscr{L}(S) \\ \Downarrow \\ \boldsymbol{\mathscr{L}(F_k) \longrightarrow 0.} \end{gathered} \tag{10}$$

We approximate the Laplacian $\mathscr{L}(X)$ at a point $x_i \in X$ as its distances from all points $x_j \in \mathcal{N}(x_i)$, where $\mathcal{N}(x_i)$ is a set of points, of size $|\mathcal{N}(x_i)|$, in a neighborhood around x_i defined in a sequel. We use L_1 norm for regularization, so that our regularization loss is:

$$\mathcal{L}_k^{reg.} = \frac{1}{n_1} \sum_{p_i\in S} \frac{1}{|\mathcal{N}(p_i)|} \sum_{p_j\in\mathcal{N}(p_i)} \|F_k(p_i) - F_k(p_j)\|_1, \tag{11}$$

where $F_k(p_i)$ is the value of the predicted scene flow at point p_i, and n_1 is the number of points in the source.

To reduce the computational overhead of nearest neighbors for a large K, we use $\mathcal{N}(p_i) = \mathcal{N}_a(p_i) \cup \mathcal{N}_b(p_i)$, where $\mathcal{N}_a(p_i)$ is the K_a nearest neighbours of p_i, and $\mathcal{N}_b(p_i)$ is calculated by random sampling K_b points in an Euclidean ball around p_i, with radius r_b.

The overall self-supervised loss is a weighted sum of Chamfer and regularization losses, over all sequence iterations:

$$\boldsymbol{\mathcal{L}^{self}} = \sum_{k=1}^{K} \mathcal{L}_k^{self} = \sum_{k=1}^{K} \alpha_k \mathcal{L}_k^{ch.} + \beta_k \mathcal{L}_k^{reg.} \tag{12}$$

5.2. Fully-supervised Loss

In order to show our architecture efficiency, we further train our system in a fully-supervised manner, using the L_1

Dataset	Method	Sup.	EPE3D↓	Acc3DS↑	AccDR↑	Outliers3D↓
FlyingThings3D	ICP[2]	*Self*	0.4062	0.1614	0.3038	0.8796
	Ego-motion[37]	*Self*	0.1696	0.2532	0.5501	0.8046
	PointPWC-Net[43]	*Self*	0.1246	0.3068	0.6552	0.7032
	Ours	***Self***	**0.0852**	**0.5363**	**0.8262**	**0.4198**
	FlowNet3D[21]	*Full*	0.1136	0.4125	0.7706	0.6016
	HPLFlowNet[12]	*Full*	0.0804	0.6144	0.8555	0.4287
	PointPWC-Net[43]	*Full*	0.0588	0.7379	0.9276	0.3424
	FLOT[31]	*Full*	0.0520	0.7320	0.9270	0.3570
	Ours	***Full***	**0.0455**	**0.8162**	**0.9614**	**0.2165**
KITTI	ICP[2]	*Self*	0.5181	0.0669	0.1667	0.8712
	Ego-motion[37]	*Self*	0.4154	0.2209	0.3721	0.8096
	PointPWC-Net[43]	*Self*	0.2549	0.2379	0.4957	0.6863
	Ours	***Self***	**0.1021**	**0.7080**	**0.8394**	**0.2456**
	FlowNet3D[21]	*Full*	0.1767	0.3738	0.6677	0.5271
	HPLFlowNet[12]	*Full*	0.1169	0.4783	0.7776	0.4103
	PointPWC-Net[43]	*Full*	0.0694	0.7281	0.8884	0.2648
	FLOT[31]	*Full*	0.0560	0.7550	0.9080	0.2420
	Ours	***Full***	**0.0546**	**0.8051**	**0.9254**	**0.1492**

Table 1. **Evaluation results on FlyingThings3D and KITTI datasets.** All methods trained only on FlyingThings3D. *Self / Full* means self-supervised / fully-supervised, where on KITTI *Self / Full* refers to the training type on FlyingThings3D of the respective model that is evaluated on KITTI. Our method outperforms all baselines on all metrics in both fully-supervised and self-supervised training. Our self-supervised version is the only self-supervised method with EPE3D below $10m$ on FlyingThings3D, and it shows its generalization ability by more than 50% improvement over existing baselines on KITTI.

loss:

$$\mathcal{L}_k^{L_1} = \frac{1}{n_1} \sum_{p_i \in S} \|F_k(p_i) - F_{GT}(p_i)\|_1, \qquad (13)$$

where $F_{GT}(p_i)$ is the value of the ground-truth scene flow at point p_i.

Unlike previous methods, we add a laplacian regularization loss to our fully-supervised training to encourage our system to preserve objects' structures and approach toward the target in iterations. The regularization loss is the same as in the self-supervised case, Eq. (11).

The overall fully-supervised loss is a weighted sum of L_1 and regularization losses, over all sequence iterations:

$$\mathcal{L}^{sv} = \sum_{k=1}^{K} \mathcal{L}_k^{sv} = \sum_{k=1}^{K} \alpha_k \mathcal{L}_k^{L_1} + \beta_k \mathcal{L}_k^{reg.}. \qquad (14)$$

6. Experiments

Following the experimental setup suggested in [12, 21, 31, 37, 43], we first train and evaluate our model on synthetic dataset FlyingThings3D [24] (Sec. 6.1) using both self-supervised and fully-supervised approaches. Then, we test the models' performance on the real-world KITTI scene flow dataset [26, 27] without any fine-tuning (Sec. 6.2). Finally, in Seq. 6.3, we conduct ablation studies regarding the inference iterations number, and the importance of the regularization loss.

Evaluation Metrics. We use the same scene flow evaluation metrics proposed by [21] and adopted by [12, 31, 43]:

- **EPE3D(m)** average end-point-error $\|F_{pred} - F_{GT}\|_2$ over each point.
- **Acc3DS(0.05)** percentage of points whose **EPE3D**$<$ $0.05m$ or relative error $< 5\%$
- **Acc3DR(0.1)** percentage of points whose **EPE3D**$<$ $0.1m$ or relative error $< 10\%$
- **Outliers3D** percentage of points whose **EPE3D**$>$ $0.3m$ or relative error $> 10\%$

6.1. Evaluation on FlyingThings3D

Due to the difficulty of acquiring dense scene flow data, we follow previous methods [12, 21, 31, 43] and train our system only on the synthetic FlyingThings3D dataset, using the same pre-processing methodology as [12].

First, we focus on a self-supervised approach, which does not require any labeled data. Then, to demonstrate our system efficiency, we also conduct experiments using a fully-supervised loss.

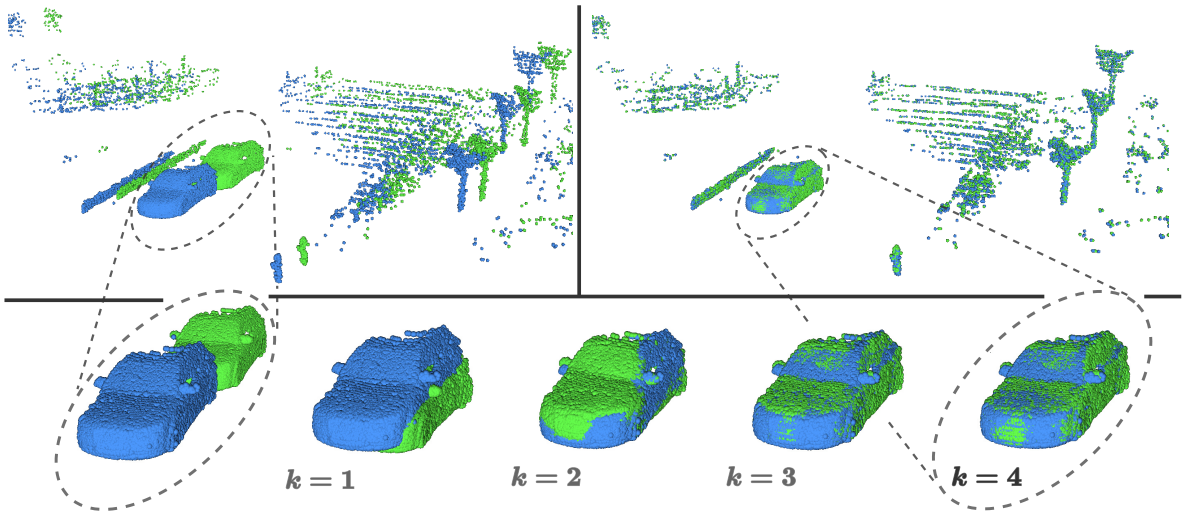

Figure 6. **Inference iterations.** Self-supervised model output example from the KITTI test set. Top-left: input source (green) and target (blue) scans. Top-right: overlay of the warped source and the target. Bottom: a closer observation on warped source toward the target during four iterations of flow estimation.

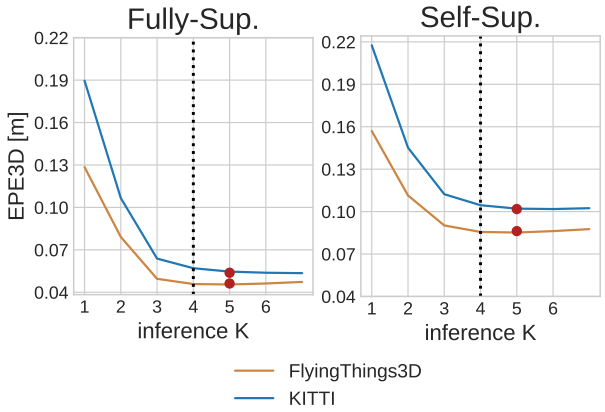

Figure 7. **EPE3D vs. inference K.** For the fully-supervised (left) and the self-supervised (right) trained models, the best K (red) based on the FlyingThings3D validation set (orange), used for test on KITTI (blue). Both models trained with $K = 4$ (dashed line).

Implementation Details. The FlyingThings3D dataset contains 19,640 pairs of point clouds in the training set and 3,824 pairs in the validation set. We first train our system on one quarter of the train data (4910 pairs) and then fine-tune on the full training set, to speed-up training. We used FlyingThings3D validation set for all hyperparameters tuning.

We use $n = 8192$ points for each point cloud, batch size of 16, and unroll $K = 4$ iterations at all training procedures, using 8 GTX-2080Ti GPUs. Pre-training is done for 90 epochs, with a learning rate of 0.002 and reduced by half at epochs $[50, 70]$. Self-supervised model is fine-tuned for 30 epochs, with a learning rate of 0.002 and reduced by half at epochs $[5, 15, 25]$. Fully-supervised model is fine-tuned for 40 epochs, with a learning rate of 0.001 and reduced by half at epochs $[10, 22, 30]$.

To reduce outliers, we limit the distance of correspondence points to a reasonable displacement range, by zeroing our coarse all-to-all correlation matrix at every entry (i, j) of which $\|p_i - q_j\|_2 > 10m$.

Lastly, we used FlyingThings3D validation set to determine the best number of iterations for our model at inference time. As discussed in Sec. 6.3, we set $K = 5$ for all tests.

All loss weights $\{\alpha_k\}_{k=1}^{K}$, $\{\beta_k\}_{k=1}^{K}$ of all training procedures, and a detailed scheme of our architecture can be found in the supplementary materials.

Results. We compare our self-supervised method's results with Iterative-Closest-Point (ICP) [2], PointPWC-Net [43], and the recent self-supervised method introduced by Tishchenko *et al.* [37], and our fully-supervised method's results with FlowNet3D [21], HPLFlowNet [12], PointPWC-Net [43], and FLOT [31].

As shown in Table 1, our method outperforms all existing methods on all evaluation metrics on the FlyingThings3D dataset, for both self-supervised and fully-supervised frameworks. Moreover, our self-supervised method is the only self-supervised method with EPE3D below $10m$ on the FlyingThings3D dataset.

6.2. Generalization on KITTI

To examine the generalization ability of our method to real-world data, we evaluate a model trained using FlyingThings3D, on real-scans KITTI Scene Flow 2015 [26, 27] dataset, without any fine-tuning. Following [12, 43], we evaluate our model on all 142 scenes with available 3D

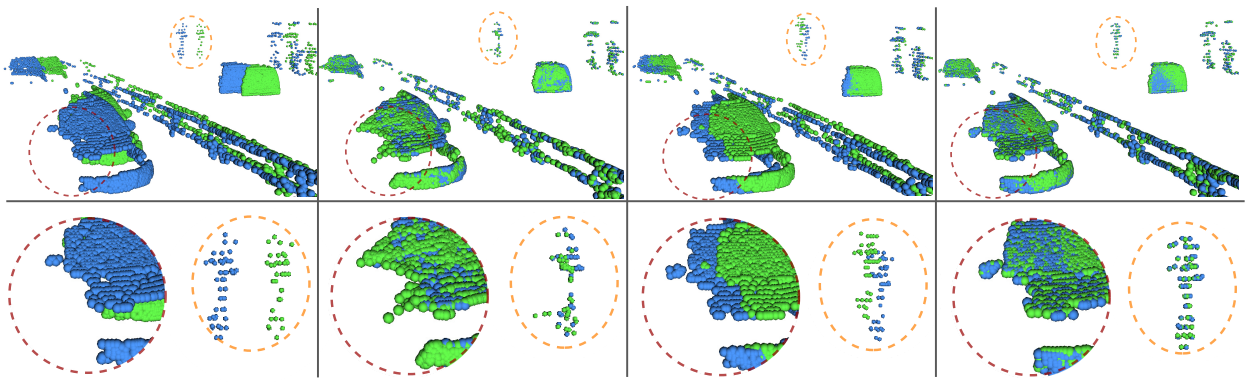

Figure 8. **Regularization inference example.** An Inference of our self-supervised model pre-trained with three different regularization schemes. From left to right: input source (green) and target (blue) scans, *under*-regularization, *over*-regularization, and our *chosen*-regularization. Interesting artifacts are circled and zoomed-in at the bottom. All variations evaluated with $K = 4$.

Regularization $(\{\alpha_k\}, \{\beta_k\})^*$	EPE3D↓	Outliers3D↓
Under-regularization	0.3183	0.7698
Over-regularization	0.2706	0.8941
***Chosen*-regularization**	**0.1443**	**0.3736**

Table 2. **Regularization.** EPE3D and outliers rate of our self-supervised model pre-trained with different regularization loss weights $\{\alpha_k\}, \{\beta_k\}$. All evaluated with $K = 4$. It can be seen that both *over*-regularization and *under*-regularization increase errors. *Exact values of all β_k, α_k are in our supplementary.

data in the training set, and remove the ground points from the point clouds by height ($< 0.3m$).

Our self-supervised method demonstrates a great generalization ability, outperforming all existing self-supervised methods by more than 50%.

Our fully-supervised model achieves EPE3D on par with state of the art method [31], highest accuracy, lowest outliers and benefits from memory efficiency (Fig. 2).

6.3. Ablation Studies

Number of iterations. Although we unrolled four iterations for training, we tested inference with different K values (Fig. 7). Interestingly, both our models keep slightly improving for a few more iterations on the KITTI test set. On the FlyingThings3D validation set, the models are optimized to one iteration more than the number of training iterations. To decrease run time, one can choose to optimize the system to a smaller number of iterations and get quite good results for 2 or 3 iterations as well. However, a model trained with K=4 not only produces the best results but also benefits from the best generalization ability, especially under large deformations. Fig. 6 shows a qualitative example of our self-supervised method during four inference iterations.

Update unit design choices. Using a GRU showed better performance than a simple fully-connected layer, increasing the system's generalization ability by 40%. Regarding its inputs, we found that using the *flow_embedding* alone as the GRU input increases the validation error by 30%.

Regularization. Since our method re-encodes the warped source at every iteration, it is crucial to train it using a regularization loss. While training with under-regularization may distort the objects' structure, over-regularization may lead to semi-rigid motion predictions, which results in imperfect alignment. To demonstrate the importance of wisely choosing the regularization loss weights, we pre-train our self-supervised model in three different regularization schemes, changing only the loss weights $\{\beta_k\}_{k=1}^K$, $\{\alpha_k\}_{k=1}^K$, and then evaluate each one of them on the KITTI test set. We show the quantitative results in Table 2, and a qualitative example in Fig. 8.

7. Conclusions

In this work, we proposed and studied a novel approach for scene flow estimation by unrolling an iterative scheme using a recurrent architecture that learns the optimal steps toward the solution, called FlowStep3D. We showed the benefit of approaching the solution in a few steps by enforcing strong regularization and re-encoding the warped scene, which is contrary to all previous learning-based solutions. Experiments performed on synthetic and real LiDAR scans data showed great generalization capability, especially for self-supervised training, improving previous methods by a large margin.

Acknowledgment. This work was partially funded by the Zimin Institute for Engineering Solutions Advancing Better Lives, the Israeli consortiums for soft robotics and autonomous driving, and the Nicholas and Elizabeth Slezak Super Center for Cardiac Research and Biomedical Engineering at Tel Aviv University.

References

[1] B. Amberg, S. Romdhani, and T. Vetter. Optimal step nonrigid icp algorithms for surface registration. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[2] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[3] Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):500–513, 2010.

[4] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[5] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[6] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. Rigid scene flow for 3d lidar scans. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1765–1770. IEEE, 2016.

[7] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[8] Nariman Farsad, Nir Shlezinger, Andrea J Goldsmith, and Yonina C Eldar. Data-driven symbol detection via model-based machine learning. *arXiv preprint arXiv:2002.07806*, 2020.

[9] J. Feldmar and N. Ayache. Rigid, affine and locally affine registration of free-form surfaces. *International Journal of Computer Vision*, 18:99–119, 2004.

[10] V. Golyanik, K. Kim, R. Maier, M. Niessner, D. Stricker, and J. Kautz. Multiframe scene flow with piecewise rigid motion. In *International Conference on 3D Vision (3DV)*, Qingdao, China, October 2017.

[11] Thibault Groueix, M. Fisher, V. Kim, Bryan C. Russell, and Mathieu Aubry. 3d-coded: 3d correspondences by deep deformation. In *ECCV*, 2018.

[12] Xiuye Gu, Y. Wang, Chongruo Wu, Y. Lee, and Panqu Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3249–3258, 2019.

[13] S. Hadfield and R. Bowden. Kinecting the dots: Particle based scene flow from depth sensors. In *2011 International Conference on Computer Vision*, pages 2290–2295, 2011.

[14] Berthold KP Horn and Brian G Schunck. Determining optical flow. In *Techniques and Applications of Image Understanding*, volume 281, pages 319–331. International Society for Optics and Photonics, 1981.

[15] Qi-Xing Huang, Bart Adams, Martin Wicke, and Leonidas J Guibas. Non-rigid registration under isometric deformations. In *Proceedings of the Symposium on Geometry Processing*, pages 1449–1457, 2008.

[16] F. Huguet and F. Devernay. A variational method for scene flow estimation from stereo sequences. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–7, 2007.

[17] V. Jampani, Martin Kiefel, and P. Gehler. Learning sparse high dimensional filters: Image filtering, dense crfs and bilateral neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4452–4461, 2016.

[18] Martin Kiefel, Varun Jampani, and Peter V Gehler. Permutohedral lattice cnns. *arXiv preprint arXiv:1412.6618*, 2014.

[19] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[20] Y. Li, M. Tofighi, V. Monga, and Y. C. Eldar. An algorithm unrolling approach to deep image deblurring. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7675–7679, 2019.

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

[22] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.

[23] Jiayi Ma, Ji Zhao, and Alan L Yuille. Non-rigid point set registration by preserving global and local structures. *IEEE Transactions on image Processing*, 25(1):53–64, 2015.

[24] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[25] Tim Meinhardt, Michael Moller, Caner Hazirbas, and Daniel Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1781–1790, 2017.

[26] Moritz Menze, Christian Heipke, and Andreas Geiger. Joint 3d estimation of vehicles and scene flow. In *Proc. of the ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015.

[27] Moritz Menze, Christian Heipke, and Andreas Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2018.

[28] Chris Metzler, Ali Mousavi, and Richard Baraniuk. Learned d-amp: Principled neural network based compressive image recovery. In *Advances in Neural Information Processing Systems*, pages 1772–1783, 2017.

[29] Himangi Mittal, Brian Okorn, and David Held. Just go with the flow: Self-supervised scene flow estimation. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11174–11182, 2020.

[30] Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *arXiv preprint arXiv:1912.10557*, 2019.

[31] Gilles Puy, Alexandre Boulch, and Renaud Marlet. Flot: Scene flow on point clouds guided by optimal transport. *ArXiv*, abs/2007.11142, 2020.

[32] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[33] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[34] Olga Sorkine. Laplacian Mesh Processing. In Yiorgos Chrysanthou and Marcus Magnor, editors, *Eurographics 2005 - State of the Art Reports*. The Eurographics Association, 2005.

[35] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2530–2539, 2018.

[36] Zachary Teed and Jun Deng. Raft: Recurrent all-pairs field transforms for optical flow. *ArXiv*, abs/2003.12039, 2020.

[37] Ivan Tishchenko, S. Lombardi, M. Oswald, and M. Pollefeys. Self-supervised learning of non-rigid residual flow and ego-motion. *ArXiv*, abs/2009.10467, 2020.

[38] Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. Three-dimensional scene flow. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 722–729. IEEE, 1999.

[39] Christoph Vogel, Konrad Schindler, and Stefan Roth. Piecewise rigid scene flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1377–1384, 2013.

[40] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–67, 2018.

[41] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the IEEE international conference on computer vision*, pages 1385–1392, 2013.

[42] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[43] 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.

[44] Y. Yang, J. Sun, H. Li, and Z. Xu. Admm-csnet: A deep learning approach for image compressive sensing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(3):521–538, 2020.

[45] K. Zhang, L. Gool, and R. Timofte. Deep unfolding network for image super-resolution. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3214–3223, 2020.