RAFT-3D: Scene Flow using Rigid-Motion Embeddings

Zachary Teed and Jia Deng
Princeton University
{zteed,jiadeng}@cs.princeton.edu

Abstract

We address the problem of scene flow: given a pair of stereo or RGB-D video frames, estimate pixelwise 3D motion. We introduce RAFT-3D, a new deep architecture for scene flow. RAFT-3D is based on the RAFT model developed for optical flow but iteratively updates a dense field of pixelwise SE3 motion instead of 2D motion. A key innovation of RAFT-3D is rigid-motion embeddings, which represent a soft grouping of pixels into rigid objects. Integral to rigid-motion embeddings is Dense-SE3, a differentiable layer that enforces geometric consistency of the embeddings. Experiments show that RAFT-3D achieves state-of-the-art performance. On FlyingThings3D, under the two-view evaluation, we improved the best published accuracy ($\delta < 0.05$) from 34.3% to 83.7%. On KITTI, we achieve an error of 5.77, outperforming the best published method (6.31), despite using no object instance supervision. Code is available at https://github.com/princeton-vl/RAFT-3D.

1. Introduction

Scene flow is the task of estimating pixelwise 3D motion between a pair of video frames[44]. Detailed 3D motion is requisite for many downstream applications including path planning, collision avoidance, virtual reality, and motion modeling. In this paper, we focus on stereo scene flow and RGB-D scene flow, which address stereo video and RGB-D video respectively.

Many scenes can be well approximated as a collection of rigidly moving objects. The motion of driving scenes, for example, can be modeled as a variable number of cars, buses, and trucks. The most successful scene flow approaches have exploited this structure by decomposing a scene into its rigidly moving components[33, 46, 46, 48, 31, 2, 21, 20, 24]. This introduces a powerful prior which can be used to guide inference. While optical flow approaches typically assume piecewise smooth motion, a scene containing rigid objects will exhibit piecewise constant 3D motion fields (Fig. 1).

Recently, many works have proposed integrating deep learning into scene flow estimation pipelines. A common approach has been to use object detection[2, 4] or segmentation [2, 31, 30, 38] networks to decompose the scene into a collection of potentially rigidly moving objects. Once the scene has been segmented into its rigidly moving components, more traditional optimization can be used to fit a motion model to each of the objects. One limitation of this approach is that the networks require instance segmentations to be trained and cannot recover the motion of new unknown objects. Object detection and instance segmentation introduce non-differentiable components into the network, making end-to-end training difficult without bounding box or instance level supervision.

We introduce RAFT-3D, an end-to-end differentiable architecture which estimates pixelwise 3D motion from stereo or RGB-D video. RAFT-3D is built on top of RAFT [42], a state-of-the-art optical flow architecture that builds all-pairs correlation volumes and uses a recurrent unit to iteratively refine a 2D flow field. We retain the basic iterative structure of RAFT but introduce a number of novel designs.

The main innovation we introduce is rigid-motion embeddings, which are per-pixel vectors that represent a soft grouping of pixels into rigid objects. During inference, RAFT-3D iteratively updates the rigid-motion embeddings such that pixels with similar embeddings belong to the same rigid object and follow the same SE3 motion.

Integral to rigid-motion embeddings is *Dense-SE3*, a differentiable layer that seeks to ensure that the embeddings are geometrically meaningful. Dense-SE3 iteratively updates a dense field of per-pixel SE3 motion by performing unrolled Gauss-Newton iterations such that the per-pixel SE3 motion is geometrically consistent with the current estimates of rigid-motion embeddings and pixel correspondence. Because of Dense-SE3, the rigid-motion embeddings can be indirectly supervised from only ground truth 3D scene flow, and our approach does not need any supervision of object boxes or masks.

Fig. 1 provides an overview of our approach. RAFT-3D take a pair of RGB-D images as input. It extracts features from the input images and builds a 4D correlation volume

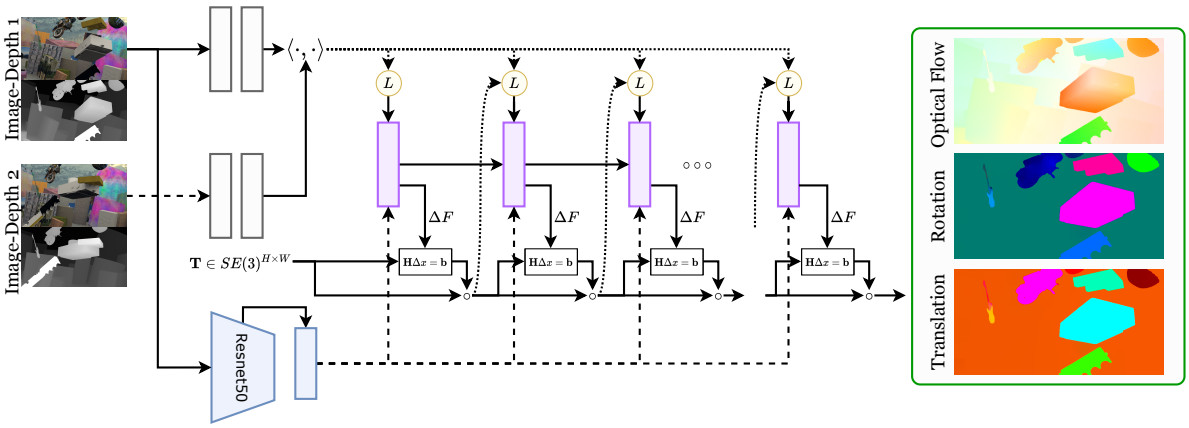

Figure 1. Overview of our approach. Features extracted from the input images are used to construct a 4D correlation volume. We initialize the SE3 motion field, $\mathbf{T}$, to be the identity at every pixel. During each iteration, the update operator uses the current SE3 motion estimate to index from the correlation volume, using the correlation features and hidden state to produce estimates of pixel correspondence and rigid-motion embeddings. These estimates are plugged into Dense-SE3, a least-squares optimization layer which uses geometric constraints to produce an update to the SE3 field. After successive iterations we recover a dense SE3 field, which can be decomposed into a rotational and translation component. The SE3 field can be projected onto the image to recover optical flow.

by computing the visual similarity between all pairs of pixels. RAFT-3D maintains and updates a dense field of pixelwise SE3 motion. During each iteration, it uses the current estimate of SE3 motion to index from the correlation volume. A recurrent GRU-based update operator takes the correlation features and produces an estimate of pixel correspondence, which is then used by Dense-SE3 to generate updates to the SE3 motion field.

RAFT-3D achieves state-of-the-art accuracy. On FlyingThings3D, under the two-view evaluation [26], RAFT-3D improves the best published accuracy ($\delta < 0.05$) from 34.3% to 83.7%. On KITTI, RAFT-3D achieves an error of 5.77, outperforming the best published method (6.31), despite using no object instance supervision.

2. Related Work

The task of reconstructing a 3D motion field from video is often referred to as estimating "scene flow".

Optical Flow: Optical flow is the problem of estimating dense 2D pixel-level motion between a pair of frames. Early work formulated optical flow as a energy minimization problem, where the objective was a combination of a data term—encouraging the matching of visually similar image regions—and a regularization term—favoring piecewise smooth motion fields. Many early scene flow approaches evolved from this formulation, replacing piecewise *smooth* flow priors with a piecewise *constant* rotation/translation field prior[47, 33]. This greater degree of structure allowed scene flow methods to outperform approaches which treated optical flow or stereo separately[46].

Recently, the problem of optical flow has been reformulated in the context of deep learning. Many works have demonstrated that a neural network can be directly trained to estimate optical flow between a pair of frames, and a large variety of network architectures have been proposed for the task [10, 19, 40, 37, 29, 50, 42]. RAFT[42] is a recurrent network architecture for estimating optical flow. RAFT builds a 4D correlation volume by computing the visual similarity between all pairs of pixels; then, during inference, a recurrent update operator indexes from the correlation volume to produce a flow update. A unique feature of RAFT is that a single, high resolution, flow field is updated and maintained.

Our approach is based on the RAFT architecture, but instead of a flow field, we estimate a SE3 motion field, where a rigid body transformation is estimated for each pixel. When projected onto the image, our SE3 motion vectors give more accurate optical flow than RAFT.

Rectified Stereo: Rectified stereo can be viewed as a 1-dimensional analog to optical flow, where the correspondence of each pixel in the left image is constrained to lie on a horizontal line spanning the right image. Like optical flow, traditional methods treated stereo as an energy minimization problem[18, 36] often exploiting planar information[3].

Recent deep learning approaches have borrowed many core concepts from conventional approaches such as the use of a 3D cost volume [23], replacing hand-crafted features and similarity metrics with learned features, and cost volume filtering with a learned 3D CNN. Like optical flow, a variety of network architectures have been proposed [23, 52, 15, 6]. Here we use GA-Net[52] to estimate depth between the each left/right image pair.

Scene Flow: Like optical flow and stereo, scene flow can be approached as a energy minimization problem. The ob-

jective is to recover a flow field such that (1) visually similar image regions are aligned and (2) the flow field maximizes some prior such as piecewise rigid motion and piecewise planar depth. Both variational optimization[35, 21, 20] and discrete optimization[33, 20] approaches have been explored for inference. Our network is designed to mimic the behavior an optimization algorithm. We maintain an estimate of the current motion field which is updated and refined with each iteration.

Jaimez et al.[21] proposed an alternating optimization approach for scene flow estimation from a pair of RGB-D images, iterating between grouping pixels into rigidly moving clusters and estimating the motion model for each of the cluster. Our method shares key ideas with this approach, namely the grouping of pixels into rigidly moving objects, however, we avoid a hard clustering by using rigid-motion embeddings, which softly and differentiably group pixels into rigid objects.

Recent works have leveraged the object detection and semantic segmentation ability of deep networks to improve scene flow accuracy[31, 4, 38, 2, 13]. In these works, an object detection or instance segmentation network is trained to identify potentially moving objects, such as cars or buses. While these approaches have been very effective for driving datasets such as KITTI where moving objects can be easily identified using semantics, they do not generalize well to novel objects. An additional limitation is that the detection and instance segmentation introduces non-differentiable components into the pipeline, requiring these components to be trained separately on ground truth annotation. Ma et al. [31] was able to train an instance segmentation network jointly with optical flow estimation by differentiating through Gauss-Newton updates; however, this required additional instance supervision and pre-training on Cityscapes[8]. On the other hand, our network outperforms these approaches without using object instance supervision.

Yang and Ramanan[51] take a unique approach and use a network to predict optical expansion, or the change in perceived object size. Combining optical expansion with optical flow gives normalized 3D scene flow. The scale ambiguity can be recovered using Lidar, stereo, or monocular depth estimation. This approach does not require instance segmentation, but also cannot directly enforce rigid motion priors.

Another line of work has focused on estimating 3D motion between a pair [26, 49, 14] or sequence[27, 11] of point clouds. These approaches are well suited for Lidar data where the sensor produces sparse measurements. However, these works do not directly exploit scene rigidity. As we demonstrate in our experiments, reasoning about object level rigidity is critical for good accuracy.

3. Approach

We propose an iterative architecture for scene flow estimation from a pair of RGB-D images. Our network takes in two image/depth pairs, (I_1, Z_1), (I_2, Z_2), and outputs a dense transformation field $\mathbf{T} \in SE(3)^{H \times W}$ which assigns a rigid body transformation to each pixel. For stereo images, the depth estimates Z_1 and Z_2 are obtained using an off-the-shelf stereo network.

3.1. Preliminaries

We use the pinhole projection model and assume known camera intrinsics. We use an augmented projection function which maps a 3D point to its projected pixel coordinates, (x, y), in addition to inverse depth $d = 1/Z$. Given a homogeneous 3D point $\mathbf{X} = (X, Y, Z, 1)$

$$(x, y, d) = \pi(\mathbf{X}) = \begin{pmatrix} f_x(X/Z) + c_x \\ f_y(Y/Z) + c_y \\ 1/Z \end{pmatrix} \tag{1}$$

where (f_x, f_y, c_x, c_y) are the camera intrinsics.

Given a dense depth map $Z \in {\mathbb{R}_+}^{H \times W}$, we can use the inverse projection function.

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \pi^{-1}(x, y, d) = \frac{1}{d} \begin{pmatrix} (x - c_x)/f_x \\ (x - c_y)/f_y \\ 1 \\ d \end{pmatrix} \tag{2}$$

which maps from pixel (x, y, d) to the point $(X, Y, Z, 1)$, again with inverse depth $d = 1/z$.

Mapping Between Images: We use a dense transformation field, $\mathbf{T} \in SE(3)^{H \times W}$ to represent the 3D motion between a pair of frames. Using $\mathbf{T}$, we can construct a function which maps points in frame I_1 to I_2. Letting $\mathbf{x}_i = (x_i, y_i, d_i)$ be the pixel coordinate at index i then the mapping

$$\mathbf{x}'_i = (x'_i, y'_i, d'_i) = \pi(\mathbf{T}_i \cdot \mathbf{X}_i), \qquad \mathbf{X}_i = \pi^{-1}(\mathbf{x}_i) \tag{3}$$

can be used to find the correspondence of $\mathbf{x}_i$ in I_2.

A flow vector can be obtained by taking the difference $\mathbf{x}'_i - \mathbf{x}_i$. The first two components of the flow vector give us the standard optical flow. The last component provides the change in inverse depth between the pair of frames. The focus of this paper is to recover $\mathbf{T}$ given a pair of frames.

Jacobians: For optimization purposes, we will need the Jacobian of the Eqn. 3. Using the chain rule, we can compute the Jacobian of Eqn. 3 as the product of the projection Jacobian

$$\mathbf{J}_\pi = \frac{\partial \pi(\mathbf{X}')}{\partial \mathbf{X}'} = \begin{pmatrix} f_x d' & 0 & -f_x X' {d'}^2 \\ 0 & f_y d' & -f_y Y' {d'}^2 \\ 0 & 0 & -{d'}^2 \end{pmatrix} \tag{4}$$

and the transformation Jacobian

$$\mathbf{J}_T = (\mathbf{I}_{3\times 3}, (\mathbf{X}')^{\wedge}), \quad \mathbf{w}^{\wedge} = \begin{pmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{pmatrix} \tag{5}$$

using local coordinates defined by the retraction $\exp(\boldsymbol{\delta}^{\wedge}) \cdot \mathbf{T}$. Giving the Jacobian of Eqn. 3 as $\mathbf{J} = \mathbf{J}_{\pi} \cdot \mathbf{J}_T \in \mathbb{R}^{3\times 6}$.

Optimization on Lie Manifolds: The space of rigid-body transformations forms a Lie group, which is a smooth manifold and a group. In this paper, we use the Gauss-Newton algorithm to perform optimization steps over the space of dense SE3 fields.

Given a weighted least squares objective

$$E(\mathbf{x}) = \sum_i w_i \cdot (f_i(\mathbf{x}) - y_i)^2 \tag{6}$$

the Gauss-Newton algorithm linearizes the residual terms, and solves for the update

$$\mathbf{J}^T \ \mathrm{diag}(\mathbf{w}) \ \mathbf{J}\Delta\mathbf{x} = \ \mathbf{J}^T\mathbf{r}(\mathbf{x}) \tag{7}$$

$$r_i = f_i(\mathbf{x}) - y_i \qquad \mathbf{J}_i = \left.\frac{\partial f_i(\exp(\delta^{\wedge})\mathbf{x})}{\partial \delta}\right|_{\delta=0} \tag{8}$$

The update is found by solving Eqn. 8 and applying the retraction $\mathbf{T}' = \exp(\Delta\mathbf{x}^{\wedge}) \cdot \mathbf{T}$. Eqn. 8 can be rewritten as the linear system

$$\mathbf{H}\Delta\mathbf{x} = \mathbf{b} \qquad \mathbf{H} = \mathbf{J}^T \ \mathrm{diag}(\mathbf{w}) \ \mathbf{J}, \ \mathbf{b} = \mathbf{J}^T\mathbf{r}(\mathbf{x}) \tag{9}$$

and $\mathbf{H}$ and $\mathbf{b}$ can be constructed without explicitly forming the Jacobian matrices

$$\mathbf{H} = \sum_i w_i \cdot \mathbf{J}_i^T \mathbf{J}_i, \qquad \mathbf{b} = \sum_i w_i \cdot \mathbf{J}_i^T r_i(\mathbf{x}). \tag{10}$$

This fact is especially useful when solving optimization functions with millions of residual terms. In this setting, storing the full Jacobian matrix becomes impractical.

3.2. Network Architecture

Our network architecture is based on RAFT[42]. We construct a full 4D correlation volume by computing the visual similarity between all pairs of pixels between the two input images. During each iteration, the network uses the current estimate of the SE3 field to index from the correlation volume. Correlation features are then fed into an recurrent update operator which estimates a dense flow field. We provide an overview of the RAFT architecture here, but more details can be found in [42].

Feature Extraction: We first extract features from the two input images. We use two separate feature extract networks. The feature encoder, f_θ, is applied to both images with shared weights. f_θ extracts a dense 128-dimension feature vector at 1/8 resolution. It consists of 6 residuals blocks, 2 at 1/2 resolution, 2 at 1/4 resolution, and 2 at 1/8 resolution. We provide more details of the network architectures in the appendix.

The context encoder extracts semantic and contextual information from the first image. Different from the original RAFT[42], we use a pretrained ResNet50[17] with a skip connection to extract context features at 1/8 resolution. The reason behind this change is that grouping objects into rigidly moving regions requires a greater degree of semantic information and larger receptive field. During training, we freeze the batch norm layers in the context encoder.

Computing Visual Similarity: We construct a 4D correlation volume by computing the dot product between all-pairs of feature vectors between the input images

$$\mathbf{C}_{ijkh}(I_1, I_2) = \langle f_\theta(I_1)_{ij}, \ f_\theta(I_2)_{kh} \rangle \in \mathbb{R}^{H\times W\times H\times W} \tag{11}$$

We then pool the last two dimensions of the correlation volume 3 times using average pooling with a 2×2 kernel, resulting in a correlation pyramid $\{\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3, \mathbf{C}_4\}$ with

$$\mathbf{C}_k \in \mathbb{R}^{H\times W\times H/2^{k-1}\times W/2^{k-1}} \tag{12}$$

Indexing the Correlation Pyramid: Given a current estimate of correspondence $\mathbf{x}' = (u, v)$, we can index from the correlation volume to produce a set of correlation features. First we construct a neighborhood grid around $\mathbf{x}$

$$\mathcal{N}_{\mathbf{x}} = \{(u + d_u, v + d_v) \mid d_u, d_v \in \{-r, ..., r\} \} \tag{13}$$

and then use the neighboorhood to sample from the correlation volume using bilinear sampling. We note that the constructing and indexing from the correlation volume is performed in an identical manner to RAFT[41].

Update Operator: The update operator is a recurrent GRU-unit which retrieves features from the correlation volume using the indexing operator and outputs a set of revisions. RAFT uses a series of 1x5 and 5x1 GRU units; we use a single 3x3 unit but use a kernel composed of 1 and 3 dilation rates. We provide more details on the architecture of the update operator in the appendix.

Using Eqn. 3, we can use the current estimate of $\mathbf{T}$ to estimate 2D correspondences $\mathbf{x}' = \pi(\mathbf{T} \cdot \pi^{-1}(\mathbf{x}))$. The following features are used as input to the GRU

- Flow field: $\mathbf{x}' - \mathbf{x}$
- Twist field: $\log_{SE3}(\mathbf{T})$
- Depth residual: $\mathbf{d}' - \bar{\mathbf{d}}'$
- Correlation features: $L_{\mathbf{C}}(\mathbf{x}')$

In the depth residual term, the inverse depth d'_i is obtained from the depth component of $\mathbf{x}'_i$, i.e. the backprojected pixel i expressed in the coordinate system of frame 2. The inverse depth $\bar{d}'_i$ is obtained by indexing the inverse depth map of frame 2 using the correspondence x'_i of pixel i. If pixel i is non-occluded, an accurate SE3 field $\mathbf{T}$ should result in a depth residual of 0. Each of the derived features are processed through 2 convolutional layers and then provided as input to the convolutional GRU.

The hidden state is then used to predict the inputs to the Dense-SE3 layer. We apply two convolutional layers to hidden state to output a rigid-motion embedding map $\mathbf{V}$. We additionally predict a "revision map" $\mathbf{r}_x, \mathbf{r}_y, \mathbf{r}_z$ and corresponding confidence maps $\mathbf{w}_x, \mathbf{w}_y, \mathbf{w}_z \in [0, 1]$. The revisions $\mathbf{r}_x$ and $\mathbf{r}_y$ correspond to corrections that should be made to the optical flow induced by the current SE3 field. In other words, the network is trying to get a new estimate of pixel correspondence, but is expressing it on top of the flow induced by the SE3 field. The revisions $\mathbf{r}_z$ is the corrections that should be made to the inverse depth in frame 2 when the inverse depth is used by Dense-SE3 to enforce geometric consistency. This is to account for noise in the input depth as well as occlusions. The embedding map and revision maps are taken as input to the Dense-SE3 layer to produce an update to the SE3 motion field.

SE3 Upsampling: The SE3 motion field estimated by the network is at 1/8 of the resolution. We use convex upsampling [42] to upsample the transformation field to the full input resolution. In RAFT[42], the high resolution flow field was taken to be the convex combination of 3×3 grids at the lower resolution with combination weights predicted by the network. However, the SE3 field $\mathbf{T}$ lies on a manifold and is not closed under linear combinations. Instead we perform upsampling by first mapping $\mathbf{T}$ to the Lie algebra using the logarithm map, performing convex upsampling in the lie algebra, and then mapping back to the manifold using the exponential map.

3.3. Dense-SE3 Layer

The key ingredient to our approach is the Dense-SE3 layer. Each application of the update module produces a revision map $\mathbf{r} = (\mathbf{r}_x, \mathbf{r}_y, \mathbf{r}_z)$. The Dense-SE3 layer is a differentiable optimization layer which maps the revision map to a SE3 field update.

The rigid-motion embedding vectors are used to softly group pixels into rigid objects. Given two embedding vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, we compute an affinity $a_{ij} \in [0, 1]$ by taking the sigmoid of the negative L2 distance

$$a_{ij} = 2 * \sigma(-||\mathbf{v}_i - \mathbf{v}_j||^2) \in [0, 1] \tag{14}$$

Objective Function: Using the affinity terms, we define an objective function based on the reprojection error

$$E(\delta) = \sum_{i \in \Omega} \sum_{j \in \mathcal{N}_i} a_{ij} e_{ij}^2(\delta_i) \tag{15}$$

$$e_{ij}^2(\delta_i) = ||\mathbf{r}_j + \pi(\mathbf{T}_j \mathbf{X}_j) - \pi(e^{\delta_i} \mathbf{T}_i \mathbf{X}_j)||_{w_j}^2 \tag{16}$$

with $||\mathbf{x}||_{\mathbf{w}}^2 = \mathbf{x}^T \operatorname{diag}(\mathbf{w}) \mathbf{x}$. The objective states that for every pixel i, we want a transformation $\mathbf{T}_i$ which describes the motion of pixels in a neighborhood $j \in \mathcal{N}_i$. However, not every pixel $j \in \mathcal{N}_i$ belongs to the same rigidly moving object. That is the purpose for the embedding vector. Only pairs (i, j) with similar embeddings significantly contribute to the objective function.

Efficient Optimization: We apply a single Gauss-Newton update to Eqn. 16 to generate the next SE3 estimate. Since the Dense-SE3 layer is applied after each application of the update operator, 12 iterations of the update operator yields 12 Gauss-Newton updates.

The objective defined in Eqn. 16 can result in a very large optimization problem. We generally use a large neighborhood $\mathcal{N}_i$ in practice; in some experiments we take $\mathcal{N}_i$ to be the entire image. For the FlyingThings3D dataset, with 540×960 resolution, this results in *200 million* equations and 50,000 variables (Dense-SE3 layer operators at 1/8 the input resolution). Trying the store the full system would exceed available memory.

However, each term in Eqn. 16 only includes a single $\mathbf{T}_i$. This means that instead of solving a single optimization problem with $H \times W \times 6$ variables, we can instead solve a set of $H \times W$ problems each with only 6 variables. Furthermore, we can leverage Eqn. 10 and build the linear system in place without explicitly constructing the Jacobian. When implemented directly in Cuda, a Gauss-Newton update of Eqn. 16 can be performed very quickly and is not a bottleneck in our approach.

3.4. bi-Laplacian Embedding Optimization

Since our architecture operates primarily at high resolution, it can be difficult for the network to group pixels which span large objects. We implement a differentiable bi-Laplacian optimization layer in order to smooth embedding vectors within motion boundaries. Vogel et al. [45] used a similar differentiable optimization layer to smooth optical flow within motion boundaries; however, they use iterative methods to solve the linear system while we use direct Cholesky factorization which allows us to reuse the factorization for each channel of the embedding vector.

Given an embedding map $\mathbf{V} \in \mathbb{R}^{H \times W \times C}$, we have the GRU predict additional edge weights $\mathbf{w}_x, \mathbf{w}_y \in \mathbb{R}_+^{H \times W}$ and define the objective

$$\mathbf{u}^* = \min_{\mathbf{u}} \left\{ ||D_x \mathbf{u}||_{\mathbf{w}_x}^2 + ||D_x \mathbf{u}||_{\mathbf{w}_y}^2 + ||\mathbf{u} - \mathbf{v}||^2 \right\} \tag{17}$$

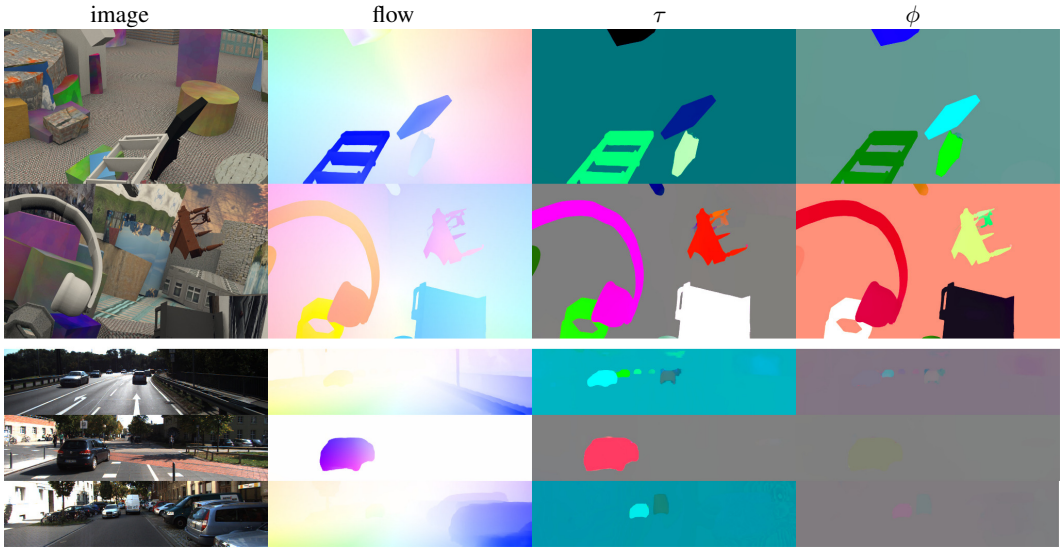

Figure 2. Visualization of the predicted motion fields on FlyingThings3D (top) and KITTI (bottom). Our network outputs a dense SE3 motion field, which can be used to compute optical flow. We visualize the SE3 field as the twist field where $(\tau, \phi) = \log_{SE3}(\mathbf{T})$. Note that the twist fields are piecewise constant—pixels from the same rigid object are assigned the same SE3 motion.

where D_x and D_y are linear finite difference operators, and $\mathbf{v}$ is the flattened feature map.

In other words, we want to solve for a new embedding map $\mathbf{u}$ which is smooth within motion boundaries and close to the original embedding map $\mathbf{v}$. At boundaries, the network can set the weights to 0 so that edges do not get smoothed over. Eqn. 17 can be solved in closed form using sparse Cholesky decomposition and we use the Cholmod library[7]. Using nested dissection[12] factorization can be performed in $O((HW)^{1.5})$ time and backsubstition can be performed in $O(C \cdot (HW)^{1.5})$ time. In the appendix, we derive the gradients of Eqn. 17. Since the optimization layer is differentiable, the inputs $\mathbf{w}_x$ and $\mathbf{w}_y$ don't require direct supervision.

3.5. Supervision

We supervise our network on a combination of ground truth optical flow and inverse depth change. Our network outputs a sequences of $\{\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_K\}$. For each transformation, $\mathbf{T}_k$, we computed the induced optical flow and inverse depth change

$$\mathbf{f}^k_{est} = \pi(\mathbf{T}_k \cdot \pi^{-1}(\mathbf{x})) - \mathbf{x} \tag{18}$$

where $\mathbf{x}$ is a dense coordinate grid in I_1. We compute the loss as the sequence over all estimations

$$\mathcal{L} = \sum_{k=1}^{N} \gamma^{N-k} ||\mathbf{f}^k_{est} - \mathbf{f}_{gt}||_1 \tag{19}$$

with $\gamma = 0.9$. Note that no supervision is applied to the embedding vectors, and that rigid-motion embeddings are implicitly learned by differentiating through the dense $SE(3)$ update layer. We also apply an additional loss directly to the revisions predicted by the GRU with 0.2 weight.

4. Experiments

We evaluate our approach on a variety of real and synthetic datasets. For all experiments we use the AdamW optimizer[28] with weight decay set to 1×10^{-5} and unroll 12 iterations of the update operator. All components of the network are trained from scratch, with the exception of the context encoder which uses ImageNet [9] pretrained weights.

Training RAFT-3D involves differentiating a computation graph which consists of both Euclidean tensors (e.g. network weights, feature activation) and Lie Groups elements (e.g. SE3 transformation field). We use the LieTorch library[43] to perform backpropagation in the tangent space of manifold elements in the computation graph.

4.1. FlyingThings3D

The FlyingThings3D dataset was introduced as part of the synthetic Scene Flow datasets by Mayer et al. [32]. The dataset consists of ShapeNet [5] shapes with randomized translation and rotations placed in a scene populated with background objects. While the dataset is not naturalistic, it offers a challenging combination of camera and object motion, each of which span all 6 degrees of freedom.

We train our network for 200k iterations with a batch size of 4 and a crop size of [320, 720]. We perform spatial augmentation by random cropping and resizing and adjust intrinsics accordingly. We use an initial learning rate of .0001 and decay the learning rate linearly during training.

Method	Input	2D Metrics		3D Metrics		
		δ_{2D} <1px	EPE	$\delta_{3D} < .05$	$\delta_{3D} < 0.10$	EPE
RAFT [42]	RGB	79.4%	3.53	-	-	-
RAFT (2D flow backprojected)	RGB-D	78.8%	3.42	50.6%	55.7%	5.442
RAFT (2D flow + depth change)	RGB-D	75.2%	3.66	33.9%	47.2%	1.218
RAFT (3D flow)	RGB-D	73.6%	4.42	36.2%	55.4%	0.266
Ours	RGB-D	**86.4%**	**2.46**	**87.8%**	**91.5%**	**0.062**

Table 1. Results on the FlyingThings3D dataset using the images from the FlowNet3D split. We evaluate on the full images (excluding pixels at infinity and extremely fast moving regions with flow > 250px)

Method	Input	$\delta_{3D} < .05$	$\delta_{3D} < 0.10$	EPE_{3D}
FlowNet3D [26]	XYZ	25.4%	57.9%	0.169
FlowNet3D++[49]	RGB-D	30.3%	63.4%	0.137
FLOT[34]	XYZ	34.3%	64.3%	0.156
Ours	RGB-D	**83.7%**	**89.2%**	**0.064**

Table 2. 3D scene flow results on the FlyingThings3D dataset using the split proposed by Liu et al [26] where only non-occluded points with depth <35m are considered for evaluation. Our method outperforms existing point-based scene flow networks by a large margin.

We evaluate our network using 2D and 3D end-point-error (EPE). 2D EPE is defined as the euclidean distance between the ground truth optical flow and the predicted optical flow which can be obtained from the 3D transformation field using Eqn. 3. 3D EPE is the euclidean distance between the ground truth 3D scene flow and the predicted scene flow. We also report threshold metrics, which measure the portion of pixels which lie within a given threshold.

In Tab. 2 we compare to point cloud based scene flow methods[26, 49, 34] using the split proposed in FlowNet3D[26] containing roughly 2000 test examples sampled from the FlyingThings3D test set. In this evaluation setup, only non-occluded pixels with depth <35 meters are used for evaluation. Our method improves the 3D $\delta < 0.05$ accuracy from 34.3% to 83.7%.

In Tab. 1 we compare to RAFT[42] and several baselines we implement to extend RAFT to predict 3D motion. All RAFT baselines use the same network architecture as our approach, including the pretrained ResNet-50. All baselines are provided with inverse depth as input which is concatenate with the input images. We also experiment with directly provided depth as input, but found that inverse depth gives the best results.

RAFT (2D flow backprojected) uses the depth maps to backproject 2D motion into a 3D flow vector, but this only works for non-occluded pixels, which is the reason for the very large 3D EPE error. RAFT (2D flow + depth change) predicts 2D flow in addition to inverse depth change, which can be used to recover 3D flow fields. Finally, we also test a version of RAFT which predicts 3D motion fields directly; RAFT(3D flow). We find that our method outperforms all these baselines by a large margin, particularly on the 3D metrics. This is because our network operates directly on the SE3 motion field, which offers a more structured representation than flow fields and we produce analytically constrained updates which the other baselines lack.

In this experiment, we evaluate over all pixels (excluding *extremely* fast moving objects with flow >250 pixels). Since we decompose the scene into rigidly moving components, our method can estimate the motion of occluded regions as well. We provide qualitative results in Fig. 2. These examples show that our network can segment the scene into rigidly moving regions, producing piecewise constant SE3 motion fields, even though no supervision is used on the embeddings.

4.2. KITTI

Using our model trained on FlyingThings3D, we finetune on KITTI for an additional 50k iterations with an initial learning rate of 5×10^{-5}. We use a crop size of [288, 960] and perform spatial and photometric augmentation. To estimate disparity, we use GA-Net[52], which provides the input depth maps for our method.

We submit our method to the KITTI leaderboard and report results from our method and other top performing methods in Tab. 3. Our approach outperforms all published methods. DRISP [31] is the next best performing approach, and combines PSMNet[6], PWC-Net[40], and Mask-RCNN[16]. Mask-RCNN is pretrained on Cityscapes and fine-tuned on KITTI using bounding box and instance mask supervision. Our network outperforms DRISP despite only training on FlyingThings3D and KITTI, and uses no instance supervision.

4.3. Ablations

We ablate various components of our model on the FlyingThings dataset and report results in Tab. 4. For all ablations, we use our network without bi-Laplacian optimization as the baseline architecture.

Iterations: We evaluate the performance of our model as function of the number of application of the update operator. We find that more iterations gives better performance up to about 16, after which we observe a slight degradation.

Methods	Runtime	Disparity 1			Disparity 2			Optical Flow			Scene Flow		
		bg	*fg*	all	*bg*	*fg*	all	*bg*	*fg*	all	*bg*	*fg*	all
OSF [33]	50 mins	4.54	12.03	5.79	5.45	19.41	7.77	5.62	18.92	7.83	7.01	26.34	10.23
SSF [38]	5 mins	3.55	8.75	4.42	4.94	17.48	7.02	5.63	14.71	7.14	7.18	24.58	10.07
Sense [22]	0.31s	2.07	3.01	2.22	4.90	10.83	5.89	7.30	9.33	7.64	8.36	15.49	9.55
DTF Sense [39]	0.76 sec	2.08	3.13	2.25	4.82	9.02	5.52	7.31	9.48	7.67	8.21	14.08	9.18
PRSM* [48]	5 mins	3.02	10.52	4.27	5.13	15.11	6.79	5.33	13.40	6.68	6.61	20.79	8.97
OpticalExp [51]	2.0 sec	**1.48**	**3.46**	**1.81**	3.39	**8.54**	4.25	5.83	**8.66**	6.30	7.06	13.44	8.12
ISF [2]	10 mins	4.12	6.17	4.46	4.88	11.34	5.95	5.40	10.29	6.22	6.58	15.63	8.08
ACOSF [25]	5mins	2.79	7.56	3.58	3.82	12.74	5.31	4.56	12.00	5.79	5.61	19.38	7.90
DRISF[31]	0.75 sec (2 GPUs)	2.16	4.49	2.55	2.90	9.73	4.04	3.59	10.40	4.73	4.39	15.94	6.31
Ours	2.0 sec	**1.48**	**3.46**	**1.81**	**2.51**	9.46	**3.67**	**3.39**	8.79	**4.29**	**4.27**	**13.27**	**5.77**

Table 3. Results of the top performing methods on the KITTI leaderboard. Ours ranks first on the leaderboard among all published methods.

Experiment	Configuration	2D Metrics		3D Metrics		
		δ_{2D} <1px	EPE	$\delta_{3D} < .05$	$\delta_{3D} < 0.10$	EPE
Iterations	1	62.1	6.05	56.0	65.9	0.212
	3	82.8	2.95	80.5	85.7	0.098
	8	85.5	2.47	86.4	90.5	0.062
	16	**85.8**	**2.43**	**87.1**	**91.0**	**0.059**
	32	85.7	2.50	87.0	90.9	0.061
Neighborhood Radius (px)	8	73.2	4.01	38.7	59.0	0.192
	64	83.8	2.52	78.1	86.6	0.078
	256	**85.8**	**2.43**	**87.1**	**91.0**	**0.059**
	Full Image	83.3	2.91	83.2	88.1	0.078
Revision Factors	Flow	**86.1**	**2.29**	84.6	88.7	0.081
	Flow + Inv. Depth	85.8	2.43	**87.1**	**91.0**	**0.059**
bi-Laplacian Smoothing	No	85.8	**2.43**	87.1	91.0	**0.059**
	Yes	**86.3**	2.45	**87.8**	**91.5**	0.062

Table 4. Ablation experiments, details of the individual experiments are provided in 4.3

Neighborhood Radius: The Dense-SE3 layer defines an objective which such at all pairs of pixels within a specific radius r contribute to the objective. Here, we train networks where r is set to $\{8, 64, 256, \infty\}$. In the last case, all pairs of pixels in the image contribute to the objective. We find that 256 gives the better performance than smaller radii; however, using the full image gives worse performance. This is likely due to the fact that most rigid objects will be less than 512 pixels in diameter, and imposing a restriction on the radius is a useful prior.

Revision Factors: The update operator produces a set of revisions which are used as input to the Dense-SE3 layer. Here we experiment with different revisions. In *Flow* we only use the optical flow revisions $\mathbf{r}_x$ and $\mathbf{r}_y$. In *flow + inv. depth* we include inverse depth revisions. We find that including inverse depth revisions leads to better performance on 3D metrics because it leverages depth consistency.

bi-Laplacian Optimization: Here we test the impact bi-Laplacian optimization layer. Our pooling layer improves the accuracy of the threshold metrics improving 1px accuracy from 85.8 to 86.3, and 3D accuracy from 87.1 to 87.8 and gives comparable average EPE. In Fig. 3 we see that the pooling layer produces qualitatively better results, particularly over large objects.

Parameter Count and Timing: RAFT-3D has 45M trainable parameters. The ResNet50 backbone has 40M parameters, while the feature extractor and update operator make up the remaining 5M parameters.

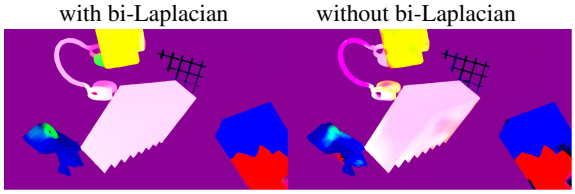

Figure 3. Impact of bi-Laplacian optimization layer on motion fields. This layer improves the ability of the network to aggregate embedding vectors within motion boundaries.

We provide a breakdown of the inference time in Tab. 5. Timing results are computed on 540x960 images with a GTX 1080Ti GPU using 16 updates. Inference on 540x960 images requires 1.6G of GPU memory, which is mainly required to store the 4D correlation volume.

Component	Time (ms)
Feature Extraction	52ms
Cost Volume	4ms
Update Operator (GRU)	208ms (13ms/iter)
Gauss Newton Iteration	120ms (7.5ms/iter)
SE3 Upsampling	2ms
Total	386ms

Table 5. Forward pass timing for different components.

5. Conclusion

We have introduced RAFT-3D, an end-to-end network for scene flow. RAFT-3D uses rigid-motion embeddings, which represent a soft grouping of pixels into rigidly moving objects. We demonstrate that these embeddings can be used to solve for dense and accurate 3D motion fields.

Acknowledgements: This research is partially supported by the National Science Foundation under Grant IIS-1942981.

References

[1] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR. org, 2017. 12

[2] Aseem Behl, Omid Hosseini Jafari, Siva Karthik Mustikovela, Hassan Abu Alhaija, Carsten Rother, and Andreas Geiger. Bounding boxes, segmentations and object coordinates: How important is recognition for 3d scene flow estimation in autonomous driving scenarios? In *International Conference on Computer Vision (ICCV)*, volume 6, 2017. 1, 3, 8

[3] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Patchmatch stereo-stereo matching with slanted support windows. In *Bmvc*, volume 11, pages 1–11, 2011. 2

[4] Zhe Cao, Abhishek Kar, Christian Hane, and Jitendra Malik. Learning independent object motion from unlabelled stereoscopic videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5594–5603, 2019. 1, 3

[5] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 6

[6] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018. 2, 7

[7] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):1–14, 2008. 6, 12

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 3

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[10] 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. 2

[11] Hehe Fan and Yi Yang. Pointrnn: Point recurrent neural network for moving point cloud processing. *arXiv preprint arXiv:1910.08287*, 2019. 3

[12] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973. 6

[13] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8977–8986, 2019. 3

[14] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3254–3263, 2019. 3

[15] Xiaoyang Guo, Kai Yang, Wukui Yang, Xiaogang Wang, and Hongsheng Li. Group-wise correlation stereo network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3273–3282, 2019. 2

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 7

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4

[18] Heiko Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 807–814. IEEE, 2005. 2

[19] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, volume 2, page 6, 2017. 2

[20] Mariano Jaimez, Mohamed Souiai, Javier Gonzalez-Jimenez, and Daniel Cremers. A primal-dual framework for real-time dense rgb-d scene flow. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 98–104. IEEE, 2015. 1, 3

[21] Mariano Jaimez, Mohamed Souiai, Jörg Stückler, Javier Gonzalez-Jimenez, and Daniel Cremers. Motion cooperation: Smooth piece-wise rigid scene flow from rgb-d images. In *2015 International Conference on 3D Vision*, pages 64–72. IEEE, 2015. 1, 3

[22] Huaizu Jiang, Deqing Sun, Varun Jampani, Zhaoyang Lv, Erik Learned-Miller, and Jan Kautz. Sense: A shared encoder network for scene-flow estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3195–3204, 2019. 8

[23] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 66–75, 2017. 2

[24] Suryansh Kumar, Yuchao Dai, and Hongdong Li. Monocular dense 3d reconstruction of a complex dynamic scene from two perspective frames. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4649–4657, 2017. 1

[25] Congcong Li, Haoyu Ma, and Qingmin Liao. Two-stage adaptive object scene flow using hybrid cnn-crf model. In *International Conference on Pattern Recognition (ICPR)*, 2020. 8

[26] 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. 2, 3, 7

[27] Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. Meteornet: Deep learning on dynamic 3d point cloud sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9246–9255, 2019. 3

[28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6

[29] Yao Lu, Jack Valmadre, Heng Wang, Juho Kannala, Mehrtash Harandi, and Philip Torr. Devon: Deformable volume network for learning optical flow. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2705–2713, 2020. 2

[30] Zhaoyang Lv, Kihwan Kim, Alejandro Troccoli, Deqing Sun, James M Rehg, and Jan Kautz. Learning rigidity in dynamic scenes with a moving camera for 3d motion field estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 468–484, 2018. 1

[31] Wei-Chiu Ma, Shenlong Wang, Rui Hu, Yuwen Xiong, and Raquel Urtasun. Deep rigid instance scene flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3614–3622, 2019. 1, 3, 7, 8

[32] 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*, pages 4040–4048, 2016. 6

[33] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3061–3070, 2015. 1, 2, 3, 8

[34] Gilles Puy, Alexandre Boulch, and Renaud Marlet. FLOT: Scene Flow on Point Clouds Guided by Optimal Transport. In *European Conference on Computer Vision*, 2020. 7

[35] Julian Quiroga, Thomas Brox, Frédéric Devernay, and James Crowley. Dense semi-rigid scene flow estimation from rgbd images. In *European Conference on Computer Vision*, pages 567–582. Springer, 2014. 3

[36] Rene Ranftl, Stefan Gehrig, Thomas Pock, and Horst Bischof. Pushing the limits of stereo using variational stereo estimation. In *2012 IEEE Intelligent Vehicles Symposium*, pages 401–407. IEEE, 2012. 2

[37] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4161–4170, 2017. 2

[38] Zhile Ren, Deqing Sun, Jan Kautz, and Erik Sudderth. Cascaded scene flow prediction using semantic segmentation. In *2017 International Conference on 3D Vision (3DV)*, pages 225–233. IEEE, 2017. 1, 3, 8

[39] René Schuster, Christian Unger, and Didier Stricker. A deep temporal fusion framework for scene flow using a learnable motion model and occlusions. *arXiv preprint arXiv:2011.01603*, 2020. 8

[40] 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. 2, 7

[41] Zachary Teed and Jia Deng. Deepv2d: Video to depth with differentiable structure from motion. In *International Conference on Learning Representations ICLR*, 2020. 4

[42] 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, 2, 4, 5, 7, 12

[43] Zachary Teed and Jia Deng. Tangent space backpropagation for 3d transformation groups. In *Conference on Computer Vision and Pattern Recognition*, 2021. 6

[44] Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. Three-dimensional scene flow. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 722–729. IEEE, 1999. 1

[45] Christoph Vogel, Patrick Knöbelreiter, and Thomas Pock. Learning energy based inpainting for optical flow. In *Asian Conference on Computer Vision*, pages 340–356. Springer, 2018. 5

[46] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a rigid motion prior. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1291–1298. IEEE, 2011. 1, 2

[47] 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. 2

[48] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015. 1, 8

[49] Zirui Wang, Shuda Li, Henry Howard-Jenkins, Victor Prisacariu, and Min Chen. Flownet3d++: Geometric losses for deep scene flow estimation. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 91–98, 2020. 3, 7

[50] Gengshan Yang and Deva Ramanan. Volumetric correspondence networks for optical flow. In *Advances in neural information processing systems*, pages 794–805, 2019. 2

[51] Gengshan Yang and Deva Ramanan. Upgrading optical flow to 3d scene flow through optical expansion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1334–1343, 2020. 3, 8

[52] 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 Conference on Computer Vision and Pattern Recognition*, pages 185–194, 2019. 2, 7

RAFT-3D: Appendix

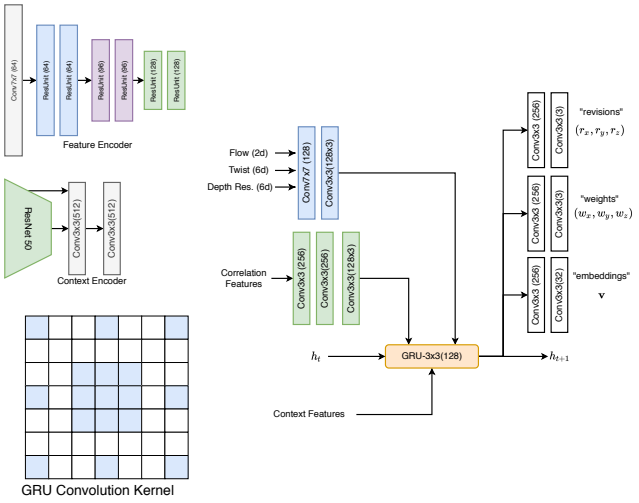

Figure 4. Network architecture. The components include (1) a feature encoder (2) a context encoder with a ResNet50 backbone, and (3) and GRU-based updated operator. The GRU uses a dilated convolution pattern as shown. In contrast to RAFT[42] where features are concatenated before being passed to the GRU, we perform elementwise addition of the context features, correlation features, and motion features.

A. Network Architecture

Details of the network architecture, including feature encoders and the GRU-based update operator are shown in Figure 4.

B. bi-Laplacian Optimization Layer Gradients

This layer minimizes an objective function in the form

$$||D_x\mathbf{u}||^2_{\mathbf{w}_x} + ||D_x\mathbf{u}||^2_{\mathbf{w}_y} + ||\mathbf{u}-\mathbf{v}||^2 \tag{20}$$

where D_x and D_y are linear finite difference operators, and $\mathbf{v}$ is the flattened feature map.

First consider the case of single channel, $\mathbf{v} \in \mathbb{R}^{HW}$. Let $W_x = \text{diag}(\mathbf{w}_x), W_y = \text{diag}(\mathbf{w}_y) \in \mathbb{R}^{HW \times HW}$. We can solve for $\mathbf{u}^*$

$$(\mathbf{I} + D_x^T W_x D_x^T + D_y^T W_y D_y^T)\mathbf{u}^* = \mathbf{v} \tag{21}$$

We perform sparse Cholesky factorization and backsubstition to solve for $\mathbf{u}^*$ using the Cholmod library[7].

Gradients: In the backward pass, given the gradient $\frac{\partial L}{\partial \mathbf{u}^*}$, we need to find the gradients with respect to the boundary weights $\frac{\partial L}{\partial \mathbf{w}_x}$ and $\frac{\partial L}{\partial \mathbf{w}_y}$.

Given the linear system $\mathbf{H}\mathbf{u} = \mathbf{v}$, the gradients with respect to $\mathbf{H}$ and $\mathbf{v}$ can be found by solving the system in the backward direction [1]

$$\frac{\partial L}{\partial \mathbf{v}} = \mathbf{H}^{-T}\frac{\partial L}{\partial \mathbf{u}^*} \tag{22}$$

$$\frac{\partial L}{\partial \mathbf{H}} = \mathbf{u}^*\mathbf{d}_v^T \tag{23}$$

$$\mathbf{d}_v = \mathbf{H}^{-T}\frac{\partial L}{\partial \mathbf{u}^*} \tag{24}$$

Here the column vector $\mathbf{d}_v$ is defined for notational convenience. Since $\mathbf{H}$ is positive definite, $\mathbf{H}^{-T} = \mathbf{H}^{-1}$ so we can reuse the factorization from the forward pass.

To compute the gradients with respect to $\mathbf{w}_x$ and $\mathbf{w}_x$

$$\frac{\partial L}{\partial \mathbf{w}_x} = \text{diag}\left(\frac{\partial L}{\partial \mathbf{H}}\frac{\partial \mathbf{H}}{\partial W_x}\right) \tag{25}$$

$$= \text{diag}\left((D_x\mathbf{u}^*)(D_x\mathbf{d}_v)^T\right) \tag{26}$$

giving

$$\frac{\partial L}{\partial \mathbf{w}_x} = (D_x\mathbf{u}^*) \odot (D_x\mathbf{d}_v) \tag{27}$$

where $\odot$ is elementwise multiplication. Similarly

$$\frac{\partial L}{\partial \mathbf{w}_y} = (D_y\mathbf{u}^*) \odot (D_y\mathbf{d}_v) \tag{28}$$

Multiple Channels: We can easily extend Eqn. 21 to work with multiple channels. Since the matrix $\mathbf{H}$ does not depend on $\mathbf{v}$, it only needs to be factored once. We can solve Eqn. 21 for all channels by reusing the factorization, treating $\mathbf{v}$ as a $HW \times C$ matrix. The gradient formulas can also be updated by summing the gradients over the channel dimensions.