HAL
open science

# FLOT: Scene Flow on Point Clouds guided by Optimal Transport

Gilles Puy, Alexandre Boulch, Renaud Marlet

## To cite this version:

Gilles Puy, Alexandre Boulch, Renaud Marlet. FLOT: Scene Flow on Point Clouds guided by Optimal Transport. European Conference on Computer Vision (ECCV), Aug 2020, Virtual conference, France. pp.527-544, 10.1007/978-3-030-58604-1_32 . hal-04327914

**HAL Id: hal-04327914**

**https://hal.science/hal-04327914**

Submitted on 6 Dec 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

arXiv:2007.11142v1 [cs.CV] 22 Jul 2020

# FLOT: Scene Flow on Point Clouds guided by Optimal Transport

Gilles Puy[1], Alexandre Boulch[1], and Renaud Marlet[1,2]

[1] valeo.ai, Paris, France
[2] ENPC, Paris, France

**Abstract.** We propose and study a method called FLOT that estimates scene flow on point clouds. We start the design of FLOT by noticing that scene flow estimation on point clouds reduces to estimating a permutation matrix in a perfect world. Inspired by recent works on graph matching, we build a method to find these correspondences by borrowing tools from optimal transport. Then, we relax the transport constraints to take into account real-world imperfections. The transport cost between two points is given by the pairwise similarity between deep features extracted by a neural network trained under full supervision using synthetic datasets. Our main finding is that FLOT can perform as well as the best existing methods on synthetic and real-world datasets while requiring much less parameters and without using multiscale analysis. Our second finding is that, on the training datasets considered, most of the performance can be explained by the learned transport cost. This yields a simpler method, $\text{FLOT}_0$, which is obtained using a particular choice of optimal transport parameters and performs nearly as well as FLOT.

## 1 Introduction

Scene flow [38] is the 3D motion of points at the surface of objects in a scene. It is one of the low level information for scene understanding, which can be useful, *e.g.*, in autonomous driving. Its estimation is a problem which has been studied for several years using different modalities as inputs such as colour images, with, *e.g.*, variational approaches [1], [45] or methods using piecewise-constant priors [16], [22], [39], or also using both colour and depth as modalities [2], [12], [32].

In this work,[3] we are interested in scene flow estimation on point clouds only using 3D point coordinates as input. In this setting, [8] proposed a technique based on the minimisation of an objective function that favours closeness of matching points for accurate scene flow estimate and local smoothness of this estimate. In [35], 2D occupancy grids are constructed from the point clouds and given as input features to a learned background removal filter and a learned classifier that find matching grid cells. A minimisation problem using these grid matches is then proposed to compute a raw scene flow before a final refinement step. In [36], a similar strategy is proposed but the match between grid cells is

[3] The code is available at `https://github.com/valeoai/FLOT`.

done using deep features. In [3], [47], the point clouds are projected onto 2D cylindrical maps and fed in a traditional CNN trained for scene flow estimation. In contrast, FLOT directly consumes point clouds by using convolutions defined for them. The closest related works are discussed in Section 2.

We split scene flow estimation into two successive steps. First, we find soft-correspondences between points of the input point clouds. Second, we exploit these correspondences to estimate the flow. Taking inspiration from recent works on graph matching that use optimal transport to match nodes/vertices in two different graphs [18], [29], [34], we study the use of such tools for finding soft-correspondences between points.

Our network takes as input two point clouds captured in the same scene at two consecutive instants $t$ and $t+1$. We extract deep features at each point using point cloud convolutions and use these features to compute a transport cost between the points at time $t$ and $t+1$. A small cost between two points indicates a likely correspondence between them. In the second step of the method, we exploit these soft-correspondences to obtain a first scene flow estimate by linear interpolation. This estimate is then refined using a residual network. The optimal transport and networks' parameters are learned by gradient descent under full supervision on synthetic datasets.

Our main contributions are: $(a)$ an optimal transport module for scene ow estimation and the study of its performance; $(b)$ a lightweight architecture that can perform as well as the best existing methods on synthetic and real-world datasets with much less parameters and without using multiscale analyses; $(c)$ a simpler method $\text{FLOT}_0$ obtained for a particular choice of the OT parameters and which achieves competing results with respect to the state-of-the-art methods. We arrive at this simplified version by noticing that most of the performance in FLOT are explained by the learned transport cost. We also notice that the main module of $\text{FLOT}_0$ can be seen as an attention mechanism. Finally, we discuss, in the conclusion, some limitations of FLOT concerning the absence of explicit treatment of occlusions in the scene.

## 2 Related Works

**Deep Scene Flow Estimation on Point Clouds.** In [4], a deep network is trained end-to-end to estimate rigid motion of objects in LIDAR scans. The closest related works where no assumption of rigidity is made are [11], [15], [40], [46]. In [40], a parametric continuous convolution that operates on data lying on irregular structures is proposed and its efficiency is demonstrated on segmentation tasks and scene flow estimation. The method [15] relies on PointNet++ [30] and uses a new flow embedding layer that learns to mix the information of both point clouds to yield accurate flow estimates. In [11], a technique to perform sparse convolutions on a permutohedral lattice is proposed. This method allows the processing of large point clouds. Furthermore, it is proposed to fuse the information of both point clouds at several scales, unlike in [15] where the information is fused once at a coarse scale. In contrast, our method fuse the information once at the finest

scale. Let us highlight that our optimal transport module is independent of the type of point cloud convolution. We choose PointNet++ but other convolution could be used. In [46], PWC-Net [33] is adapted to work on point clouds. The flow is estimated in a coarse-to-fine scale fashion showing improvement over the previous method. Finally, let us mention that recent works [25], [46] address this topic using self-supervision. We however restrict ourselves to full supervision in this work.

**Graph Matching by Optimal Transport.** Our method is inspired by recent works on graphs comparison using optimal transport. In [18], the graph Laplacian is used to map a graph to a multidimensional Gaussian distribution that represents the graph structure. The Wasserstein distance between these distributions is then used as a measure of graph similarity and permits one to match nodes between graphs. In [27], each graph is represented as a bag-of-vectors (one vector per node) and the measure of similarity is the Wasserstein distance between these sets. In [29], a method building upon the Gromov-Wasserstein distance between metric-measure spaces [21] is proposed to compare similarity matrices. This method can be used to compare two graphs by, *e.g.*, representing each of them with a matrix containing the geodesic distances between all pairs of nodes. In [34], it is proposed to compare graphs by fusing the Gromov-Wassertsein distance with the Wasserstein distance. The former is used to compare the graph structures while the latter is used to take into account node features. In our work, we use the latter distance. A graph is constructed for each point cloud by connecting each point to its nearest neighbours. We then propose a method to train a network that extract deep features for each point and use these features to match points between point clouds in our optimal transport module.

**Algorithm Unrolling.** Our method is based on the algorithm unrolling technique which consists in taking an iterative algorithm, unrolling a fixed number of its iterations, and replacing part of the matrix multiplications/convolutions in these unrolled iterations by new ones trained specifically for the task to achieve. Several works build on this technique, such as [10], [17], [24], [26] to solve linear inverse problems, or [5], [14], [20], [41] in for image denoising (where the denoiser is sometimes used to solve yet another inverse problem). In this work, we unroll few iterations of the Sinkhorn algorithm and train the cost matrix involved in it. This matrix is trained so that the resulting transport plan provides a good scene flow estimate. Let us mention that this algorithm is also unrolled, *e.g.*, in [9] to train a deep generative network, and in [31] for image feature assignments.

## 3 Method

### 3.1 Step 1: Finding Soft-Correspondences between Points

Let $\boldsymbol{p}$, $\boldsymbol{q} \in \mathbb{R}^{n\times 3}$ be two point clouds of the same scene at two consecutive instants $t$ and $t+1$. The vectors $\boldsymbol{p}_i, \boldsymbol{q}_j \in \mathbb{R}^3$ are the $xyz$ coordinates of the $i^{\text{th}}$ and $j^{\text{th}}$ points of $\boldsymbol{p}$ and $\boldsymbol{q}$, respectively. The scene flow estimation problem on point clouds consists in estimating the scene flow $\boldsymbol{f} \in \mathbb{R}^{n\times 3}$ where $\boldsymbol{f}_i \in \mathbb{R}^3$ is the translation of $\boldsymbol{p}_i$ from $t$ to $t+1$.

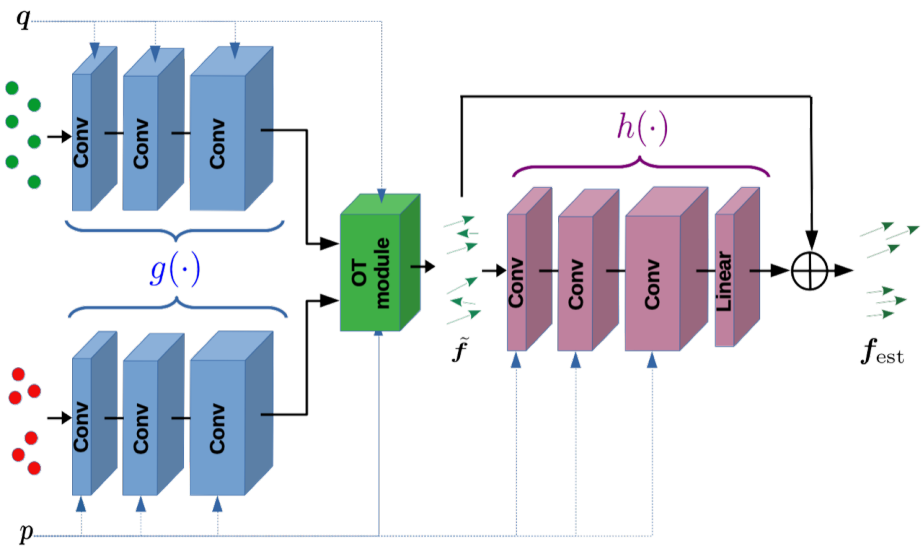


**Fig. 1.** The point clouds $\boldsymbol{p}$ and $\boldsymbol{q}$ go through $g$ which outputs a feature for each input point. These features (black arrows) go in our proposed OT module where they are used to compute the pairwise similarities between each pair of points $(\boldsymbol{p}_i, \boldsymbol{q}_j)$. The output of the OT module is a transport plan which informs us on the correspondences between the points of $\boldsymbol{p}$ and $\boldsymbol{q}$. This information permits us to compute a first scene flow estimate $\tilde{\boldsymbol{f}}$, which is refined by $h$ to obtain $\boldsymbol{f}_{\text{est}}$. The convolution layers (conv) are based on PointNet++ [30] but the OT module could accept the output of any other point cloud convolution. The dashed-blue arrows indicate that the point coordinates are passed to each layer to be able to compute convolutions on points.

**Perfect World.** We construct FLOT starting in the perfect world where $\boldsymbol{p}+\boldsymbol{f} = \mathsf{P}\,\boldsymbol{q}$, with $\mathsf{P} \in \{0,1\}^{n\times n}$ a permutation matrix. The role of FLOT is to estimate the permutation matrix $\mathsf{P}$ without the knowledge of $\boldsymbol{f}$. In order to do so, we use tools from optimal transport. We interpret the motion of the points $\boldsymbol{p}_i$ as a displacement of mass between time $t$ and $t+1$. Each point in the first point cloud $\boldsymbol{p}$ is attributed a mass which we fix to $n^{-1}$. Each point $\boldsymbol{q}_j$ then receives the mass $n^{-1}$ from $\boldsymbol{p}_i$ if $\boldsymbol{p}_i + \boldsymbol{f}_i = \boldsymbol{q}_j$, or, equivalently, if $\mathsf{P}_{ij} = 1$. We propose to estimate the permutation matrix $\mathsf{P}$ by computing a transport plan $\mathsf{T} \in \mathbb{R}_+^{n\times n}$ from $\boldsymbol{p}$ to $\boldsymbol{q}$ which satisfies

$$\mathsf{T} \in \underset{\mathsf{U}\in\mathbb{R}_+^{n\times n}}{\operatorname{argmin}} \sum_{i,j=1}^{n} \mathsf{C}_{ij}\mathsf{U}_{ij} \quad \text{subject to} \quad \mathsf{U}\mathbf{1} = \mathbf{1}n^{-1} \;\text{ and }\; \mathsf{U}^\intercal\mathbf{1} = \mathbf{1}n^{-1}, \tag{1}$$

where $\mathbf{1} \in \mathbb{R}^n$ is the vector with all entries equal to 1, and $\mathsf{C}_{ij} \geqslant 0$ is the displacement cost from point $\boldsymbol{p}_i$ to point $\boldsymbol{q}_j$ [28]. Each scalar entry $\mathsf{T}_{ij} \geqslant 0$ of the transport plan $\mathsf{T}$ represents the mass that is transported from $\boldsymbol{p}_i$ to $\boldsymbol{q}_j$.

The first constraint in (1) imposes that the mass of each point $\boldsymbol{p}_i$ is entirely distributed over some of the points in $\boldsymbol{q}$. The second constraint imposes that each

**Input:** cost matrix $\mathsf{C}$; parameters $K, \lambda, \epsilon > 0$.
**Output:** transport plan $\mathsf{T}$.
$\boldsymbol{a} \leftarrow \mathbf{1}n^{-1}$;
$\mathsf{U} \leftarrow \exp(-\mathsf{C}/\epsilon)$;
**for** $k = 1, \ldots, K$ **do**
  $\boldsymbol{b} \leftarrow [(\mathbf{1}n^{-1}) \oslash (\mathsf{U}^\intercal \boldsymbol{a})]^{\lambda/(\lambda+\epsilon)}$ ;
  $\boldsymbol{a} \leftarrow [(\mathbf{1}n^{-1}) \oslash (\mathsf{U}\, \boldsymbol{b})]^{\lambda/(\lambda+\epsilon)}$ ;
**end**
$\mathsf{T} \leftarrow \mathrm{diag}(\boldsymbol{a})\, \mathsf{U}\, \mathrm{diag}(\boldsymbol{b})$ ;

**Algorithm 1:** Optimal transport module. The symbol $\oslash$ denote the element-wise division and multiplication, respectively.

points $\boldsymbol{q}_j$ receives exactly a mass $n^{-1}$ from some of the points $\boldsymbol{p}$. No mass is lost during the transfer. Note that in the hypothetical case where the cost matrix $\mathsf{C}$ would contain one zero entry per line and per column then the transport plan is null everywhere except on these entries and the mass constraints are immediately satisfied via a simple scaling of the transport plan. In this hypothetical situation, the mass constraints would be redundant for our application as it would have been enough to find the zero entries of $\mathsf{C}$ to estimate $\mathsf{P}$. It is important to note the mass constraints play a role in the more realistic situation where "ambiguities" are present in $\mathsf{C}$ by ensuring that each point gives/receives a mass $n^{-1}$ and that each point in $\boldsymbol{p}$ has a least one corresponding point in $\boldsymbol{q}$ and vice-versa.

We note that $n^{-1}\mathsf{P}$ satisfies the optimal transport constraints. We need now to construct $\mathsf{C}$ so that $\mathsf{T} = n^{-1}\mathsf{P}$.

**Real World and Fast Estimation of $\mathsf{T}$.** In the real world, the equality $\boldsymbol{p} + \boldsymbol{f} = \mathsf{P}\,\boldsymbol{q}$ does not hold because the surfaces are not sampled at the same physical locations at $t$ and $t+1$ and because objects can (dis)appear due to occlusions. A consequence of these imperfections is that the mass preservation in (1) does not hold exactly: mass can (dis)appear. One solution to circumvent this issue is to relax the constraints in (1). Instead of solving (1), we propose to solve

$$\min_{\mathsf{U} \in \mathbb{R}_+^{n \times n}} \left[ \sum_{i,j=1}^{n} \mathsf{C}_{ij}\mathsf{U}_{ij} + \epsilon\, \mathsf{U}_{ij} \left(\log \mathsf{U}_{ij} - 1\right) \right] + \lambda\, \mathrm{KL}\left(\mathsf{U}\mathbf{1},\, \frac{\mathbf{1}}{n}\right) + \lambda\, \mathrm{KL}\left(\mathsf{U}^\intercal \mathbf{1},\, \frac{\mathbf{1}}{n}\right), \tag{2}$$

where $\epsilon, \lambda \geqslant 0$, and KL denotes the KL-divergence. The term $\mathsf{U}_{ij}(\log \mathsf{U}_{ij} - 1)$ in (2) is an entropic regularisation on the transport plan. Its main purpose, in our case, is to allow the use of an efficient algorithm to estimate the transport plan: the Sinkhorn algorithm [7]. The version of this algorithm for the optimal transport problem (2) is derived in [6] and is presented in Alg. 1. The parameter $\epsilon$ controls the amount of entropic regularisation. The smaller $\epsilon$ is, the sparser the transport plan is, hence finding sparse correspondences between $\boldsymbol{p}$ and $\boldsymbol{q}$. The regularisation parameter $\lambda$ adjust how much the transported mass deviates

from the uniform distribution, allowing mass variation. One could let $\lambda \to +\infty$ to impose strict mass preservation.

Note that the mass regularisation is controlled by the power $\lambda/(\lambda+\epsilon)$ in Alg. 1. This power tends to 1 when $\lambda \to +\infty$ to impose strict mass preservation and reaches 0 in absence of any regularisation. Instead of fixing the parameters $\epsilon, \lambda$ in advance, we let these parameters free and learn them by gradient descent along with the other networks' parameters.

We would like to recall that, in the perfect world, it is not necessary for the power $\lambda/(\lambda+\epsilon)$ to reach 1 to yield accurate results as the final quality is also driven by the quality of $\mathsf{C}$. In a perfect situation where the cost would be perfectly trained with a bijective mapping already encoded in $\mathsf{C}$ by its zero entries, then any amount of mass regularisation is sufficient to reach accurate results. This follows from our remark at the end of the previous subsection but also from the discussion in the subsection below on the role of $\mathsf{C}$ and the mass regularisation. In a real situation, the cost is not perfectly trained and we expect the power $\lambda/(\lambda+\epsilon)$ to vary in the range of $(0,1)$ after training, reaching values closer to 1 when trained in a perfect world setting and closer to 0 in presence of occlusions.

**Learning the Transport Cost.** An essential ingredient in (2) is the cost $\mathsf{C} \in \mathbb{R}^{n\times n}$ where each entry $\mathsf{C}_{ij}$ encodes the similarity between $\boldsymbol{p}_i$ to point $\boldsymbol{q}_j$. An obvious choice could be to take the Euclidean distance between each pair of points $(\boldsymbol{p}_i, \boldsymbol{q}_j)$, *i.e.*, $\mathsf{C}_{ij} = \|\boldsymbol{p}_i - \boldsymbol{q}_j\|_2$, but this choice does not yield accurate results. In this work, we propose to learn the displacement costs by training a deep neural network $g : \mathbb{R}^{n\times 3} \to \mathbb{R}^{n\times c}$ that takes as input a point cloud and output a feature of size $c$ for each input point. The entries of the cost matrix are then defined using the cosine distance between the features $g(\boldsymbol{p})_i, g(\boldsymbol{q})_j \in \mathbb{R}^c$ at points $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$, respectively:

$$\mathsf{C}_{ij} = \left(1 - \frac{g(\boldsymbol{p})_i^\intercal \; g(\boldsymbol{q})_j}{\|g(\boldsymbol{p})_i\|_2 \, \|g(\boldsymbol{q})_j\|_2}\right) \cdot i_{\|\cdot\|_2 \leqslant d_{\max}} \left(\boldsymbol{p}_i - \boldsymbol{q}_j\right). \tag{3}$$

The more similar the features $g(\boldsymbol{p})_i$ and $g(\boldsymbol{q})_j$ are, the less the cost of transporting a unit mass from $\boldsymbol{p}_i$ to $\boldsymbol{q}_j$ is. The indicator function

$$i_{\|\cdot\|_2 \leqslant d_{\max}} \left(\boldsymbol{p}_i - \boldsymbol{q}_j\right) = \begin{cases} 1 \text{ if } \|\boldsymbol{p}_i - \boldsymbol{q}_j\|_2 \leqslant d_{\max}, \\ +\infty \text{ otherwise,} \end{cases} \tag{4}$$

is used to prevent the algorithm to find correspondences between points too far away from each other. We set $d_{\max} = 10$ m.

In order to train the network $g$, we adopt the same strategy as, *e.g.*, in [9] to train generative models or in [31] for matching image features. The strategy consists in unrolling $K$ iterations of Alg. 1. This unrolled iterations constitute our OT module in Fig. 1. One can remark that the gradients can backpropagate through each step of this module and allow us to train $g$.

**On the Role of $\mathsf{C}$ and of the Mass Regularisation.** We gather in this paragraph the earlier discussions on the role of $\mathsf{C}$ and the mass regularisation.

For the sake of the explanation, we come back in the perfect-world setting and consider (1). In this ideal situation, one could further dream that it is possible to train $g$ perfectly such that $\mathsf{C}_{ij}$ is null for matching points, *i.e.*, when $\mathsf{P}_{ij} = 1$, and strictly positive otherwise. The transport plan would then satisfy $\mathsf{T} = n^{-1}\mathsf{P}$ with a null transport cost. However, one should note that the solution $\mathsf{T}$ would entirely be encoded in $\mathsf{C}$ up to a global scaling factor: the non-zero entries of $\mathsf{T}$ are at the zero entries of $\mathsf{C}$. In that case, the mass transport constraints only adjust the scale of the entries in $\mathsf{T}$. Such a perfect scenario is unlikely to occur but these considerations highlight that the cost matrix $\mathsf{C}$ could be exploited alone and could maybe be sufficient to find the appropriate correspondences between $\boldsymbol{p}$ and $\boldsymbol{q}$ for scene flow estimation. The mass transport regularisation plays a role in the more realistic case where ambiguities appears in $\mathsf{C}$. The regularisation enforces, whatever the quality of $\mathsf{C}$ and with a "strength" controlled by $\lambda$, that the mass is distributed as uniformly as possible over all points. This avoids that some points in $\boldsymbol{p}$ are left with no matching point in $\boldsymbol{q}$, and vice-versa.

**FLOT$_0$.** FLOT$_0$ is a version of FLOT where only the cost matrix $\mathsf{C}$ is exploited to find correspondences between $\boldsymbol{p}$ and $\boldsymbol{q}$. This method is obtained when removing the mass transport regularisation in (2), *i.e.*, by setting $\lambda = 0$. In this limit, the "transport plan" satisfies

$$\mathsf{T} = \exp(-\,\mathsf{C}/\epsilon). \tag{5}$$

$\mathsf{T}$ is then used in the rest of the method as if it was the output of Alg. 1.

### 3.2 Step 2: Flow Estimation from Soft-Correspondences

We obtained, in the previous step, a transport plan $\mathsf{T}$ that gives correspondences between the points of $\boldsymbol{p}, \boldsymbol{q}$. Our goal now is to exploit these correspondences to estimate the flow. As before, it is convenient to start in the perfect world and consider (1). In this setting, we have seen that $\boldsymbol{f} = \mathsf{P}\boldsymbol{q} - \boldsymbol{p}$ and that, if $g$ is well trained, we expect $n^{-1}\mathsf{P} = \mathsf{T}$. Therefore, an obvious estimate of the flow is

$$\tilde{\boldsymbol{f}}_i = \sum_{j=1}^{n} \mathsf{P}_{ij}\, \boldsymbol{q}_j - \boldsymbol{p}_i = \frac{1}{n^{-1}} \sum_{j=1}^{n} \mathsf{T}_{ij}\, \boldsymbol{q}_j - \boldsymbol{p}_i = \frac{\sum_{j=1}^{n} \mathsf{T}_{ij}\, \boldsymbol{q}_j}{\sum_{j=1}^{n} \mathsf{T}_{ij}} - \boldsymbol{p}_i, \tag{6}$$

where we exploited the fact that $\sum_{j=1}^{n} \mathsf{T}_{ij} = n^{-1}$ in the last equality.

In the real world, the first equality in (6) does not hold. Yet, the last expression in (6) remains a sensible first estimation of the flow. Indeed, this computation is equivalent to computing, for each point $\boldsymbol{p}_i$, a corresponding virtual point that is a barycentre of some points in $\boldsymbol{q}$. The larger the transported mass $\mathsf{T}_{ij}$ from $\boldsymbol{p}_i$ to $\boldsymbol{q}_j$ is, the larger the contribution of $\boldsymbol{q}_j$ to this virtual point is. The difference between this virtual point and $\boldsymbol{p}_i$ gives an estimate of the flow $\boldsymbol{f}_i$. This virtual point is a "guess" on the location of $\boldsymbol{p}_i + \boldsymbol{f}_i$ made knowing where the mass from $\boldsymbol{p}_i$ is transported in $\boldsymbol{q}$.

However, we remark that the flow $\tilde{\boldsymbol{f}}$ estimated in (6) is, necessarily, still imperfect as it is highly likely that some points in $\boldsymbol{p}+\boldsymbol{f}$ cannot be expressed as barycentres of the found corresponding points $\boldsymbol{q}$. Indeed, some portion of objects visible in $\boldsymbol{p}$ might not visible any more in $\boldsymbol{q}$ due to the finite resolution in point cloud sampling. The flow in these missing regions cannot be reconstructed from $\boldsymbol{q}$ but has to be reconstructed using structural information available in $\boldsymbol{p}$, relying on neighbouring information from the well sampled regions. Therefore, we refine the flow using a residual network:

$$\boldsymbol{f}_{\text{est}} = \tilde{\boldsymbol{f}} + h(\tilde{\boldsymbol{f}}), \tag{7}$$

where $h : \mathbb{R}^{n\times 3} \to \mathbb{R}^{n\times c}$ takes as inputs the estimated flow $\tilde{\boldsymbol{f}}$ and uses convolutions defined on the point cloud $\boldsymbol{p}$.

Let us finally conclude this section by highlighting the fact that, in the case of $\text{FLOT}_0$, (6) simplifies to

$$\tilde{\boldsymbol{f}}_i = \frac{\sum_{j=1}^{n} \exp(-\,\mathsf{C}_{ij}/\epsilon)\,(\boldsymbol{q}_j - \boldsymbol{p}_i)}{\sum_{j=1}^{n} \exp(-\,\mathsf{C}_{ij}/\epsilon)}. \tag{8}$$

On can remark that the OT module essentially reduces to an attention mechanism [37] in that case. The attention mechanism is thus a particular case of FLOT where the entropic regularisation $\epsilon$ plays the role of the softmax temperature. Let us mention that similar attention layers haved been showed effective in related problems such as rigid registration [42,43,44].

### 3.3 Training

The network's parameters, denoted by $\theta$, and $\epsilon, \gamma$ are trained jointly under full supervision on annotated synthetic datasets of size $L$. Note that to enforce positivity of $\epsilon, \gamma$, we learn their log values. A constant offset of 0.03 is applied to $\epsilon$ to avoid numerical instabilities in the exponential function during training.

The sole training loss is the $\ell_1$-norm between the ground truth flow $\boldsymbol{f}$ and the estimated flow $\boldsymbol{f}_{\text{est}}$:

$$\min_{\theta} \frac{1}{3L} \sum_{l=1}^{L} \left\| \mathsf{M}^{(l)} \left( \boldsymbol{f}_{\text{est}}^{(\ell)} - \boldsymbol{f}^{(\ell)} \right) \right\|_1, \tag{9}$$

where $\mathsf{M}^{(l)} \in \mathbb{R}^{n\times n}$ is a diagonal matrix encoding an annotated mask used to remove points where the flow is occluded.

We use a batchsize of 4 at $n = 2048$ and a batchsize of 1 at $n = 8192$ using Adam [13] and a starting learning rate of 0.001. The learning rate is kept constant unless specified in Section 4.

### 3.4 Similarities and Differences with Existing Techniques

A first main difference between FLOT and [11], [15], [46] is the number of parameters which is much smaller for FLOT (see Table 1). Another difference is

that we do not use any downsampling and upsampling layers. Unlike [11], [46], we do not use any multiscale analysis to find the correspondences between points. The information between point clouds is mixed only once, as in [15], but at the finest sampling resolution and without using skip connections between $g$ and $h$.

We also notice that [11], [15], [46] rely on a MLP or a convnet applied on the concatenated input features to mix the information between both point clouds. The mixing function is learned and thus not explicit. It is harder to find how the correspondences are effectively done, *i.e.*, identify what input information is kept or not taken into consideration. In contrast, the mixing function in FLOT is explicit with only two scalars $\epsilon, \lambda$ adjusted to the training data and whose roles are clearly identified in the OT problem (2). The core of the OT module is a simple cross-correlation between input features, which is a module easy to interpret, study and visualise. Finally, among all the functions that the convnets/MLPs in [11], [15], [46] can approximate, it is unlikely that the resulting mixing function actually approximates the Sinkhorn algorithm, or an attention layer, after learning without further guidance than those of the training data.

## 4 Experiments

### 4.1 Datasets

As in related works, we train our network under full supervision using FlyingThings3D [19] and test it on FlyingThings3D and KITTI Scene Flow [22,23]. However, none of the datasets provide point clouds directly. This information needs to be extracted from the original data. There is at least two slightly different ways of extracting these 3D data, and we report results for both versions for a better assessment of the performance. The first version of the datasets are prepared[4] as in [11]. No occluded point remains in the processed point clouds. We denote these datasets $\text{FT3D}_\text{s}$ and $\text{KITTI}_\text{s}$. The second version of the datasets are the ones prepared[5] by [15] and denoted $\text{FT3D}_\text{o}$ and $\text{KITTI}_\text{o}$. These datasets contains points where the flow is occluded. These points are present at the input and output of the networks but are not taken into account to compute the training loss (9) nor the performance metrics, like in [15]. Further information about the datasets is in the supplementary material. Note that we keep aside 2000 examples from the original training sets of $\text{FT3D}_\text{s}$ and $\text{FT3D}_\text{o}$ as validation sets, which are used in Section 4.3.

### 4.2 Performance Metrics

We use the four metrics adopted in [11], [15], [46]: the end point error EPE; two measures of accuracy, denoted by AS and AR, computed with different thresholds on the EPE; a percentage of outliers also computed using a threshold on the EPE. The definition of these metrics is recalled in the supplementary material.

[4] Code and pretrained model available at `https://github.com/laoreja/HPLFlowNet`.
[5] Code and datasets available at `https://github.com/xingyul/flownet3d`.

**Table 1.** Performance of FLOT on the validation sets of $\text{FT3D}_\text{p}$, $\text{FT3D}_\text{s}$, and $\text{FT3D}_\text{o}$ (top). Performance of FLOT measured at the output of the OT module, *i.e.*, before refinement by $h$, on $\text{FT3D}_\text{p}$ and $\text{FT3D}_\text{s}$ (bottom). The corresponding performance on $\text{FT3D}_\text{o}$ is in the supplementary material. We report average scores and, between parentheses, their standard deviations. Please refer to Section 4.3 for more details.

| | Dataset | K | $\epsilon$ | $\lambda/(\lambda+\epsilon)$ | EPE | AS | AR | Out. |
|---|---|---|---|---|---|---|---|---|
| **With flow refinement** | $\text{FT3D}_\text{p}$ | $\text{FLOT}_0$ | 0.03 (0.00) | 0 (fixed) | 0.0026 (0.0005) | 99.56 (0.08) | 99.69 (0.05) | 0.44 (0.10) |
| | | 1 | 0.03 (0.00) | 0.70 (0.00) | 0.0011 (0.0001) | 99.83 (0.01) | 99.89 (0.01) | 0.17 (0.01) |
| | | 3 | 0.03 (0.00) | 0.82 (0.00) | 0.0009 (0.0001) | 99.85 (0.01) | 99.90 (0.01) | 0.16 (0.01) |
| | | 5 | 0.03 (0.00) | 0.88 (0.00) | 0.0009 (0.0001) | 99.84 (0.02) | 99.90 (0.01) | 0.17 (0.02) |
| | $\text{FT3D}_\text{s}$ | $\text{FLOT}_0$ | 0.03 (0.00) | 0 (fixed) | 0.0811 (0.0005) | 50.32 (0.34) | 83.08 (0.24) | 52.15 (0.34) |
| | | 1 | 0.03 (0.00) | 0.64 (0.01) | 0.0785 (0.0003) | 50.91 (0.52) | 83.67 (0.10) | 51.73 (0.38) |
| | | 3 | 0.03 (0.00) | 0.59 (0.00) | 0.0786 (0.0010) | 51.06 (0.95) | 83.78 (0.35) | 51.72 (0.76) |
| | | 5 | 0.03 (0.00) | 0.56 (0.00) | 0.0798 (0.0003) | 49.77 (0.50) | 83.39 (0.08) | 52.58 (0.31) |
| | $\text{FT3D}_\text{o}$ | $\text{FLOT}_0$ | 0.03 (0.00) | 0 (fixed) | 0.1834 (0.0018) | 21.94 (0.69) | 52.79 (0.53) | 77.19 (0.43) |
| | | 1 | 0.03 (0.00) | 0.50 (0.01) | 0.1798 (0.0009) | 22.01 (0.14) | 53.39 (0.24) | 76.77 (0.16) |
| | | 3 | 0.03 (0.00) | 0.34 (0.00) | 0.1797 (0.0014) | 22.77 (0.53) | 53.74 (0.54) | 76.39 (0.43) |
| | | 5 | 0.03 (0.00) | 0.35 (0.01) | 0.1813 (0.0020) | 22.64 (0.41) | 53.58 (0.41) | 76.52 (0.46) |
| **No flow refinement** | $\text{FT3D}_\text{p}$ | $\text{FLOT}_0$ | *Same as above* | | 0.0026 (0.0006) | 99.59 (0.07) | 99.70 (0.05) | 0.42 (0.10) |
| | | 1 | | | 0.0010 (0.0001) | 99.83 (0.01) | 99.89 (0.01) | 0.18 (0.01) |
| | | 3 | | | 0.0009 (0.0000) | 99.85 (0.01) | 99.90 (0.01) | 0.16 (0.01) |
| | | 5 | | | 0.0010 (0.0001) | 99.84 (0.03) | 99.90 (0.01) | 0.17 (0.02) |
| | $\text{FT3D}_\text{s}$ | $\text{FLOT}_0$ | *Same as above* | | 0.1789 (0.0004) | 17.57 (0.07) | 43.34 (0.08) | 75.34 (0.07) |
| | | 1 | | | 0.1721 (0.0005) | 18.24 (0.11) | 44.64 (0.14) | 74.54 (0.11) |
| | | 3 | | | 0.1764 (0.0003) | 17.64 (0.07) | 43.52 (0.10) | 75.09 (0.07) |
| | | 5 | | | 0.1761 (0.0009) | 17.68 (0.13) | 43.60 (0.23) | 75.07 (0.13) |

Let us highlight that the performance reported on $\text{KITTI}_\text{s}$ and $\text{KITTI}_\text{o}$ are obtained by using the model trained on $\text{FT3D}_\text{s}$ and $\text{FT3D}_\text{o}$, respectively without fine tuning. We do not adapt the model for any of the method. We nevertheless make sure that the $xyz$ axes are in correspondence for all datasets.

### 4.3 Study of FLOT

We use $\text{FT3D}_\text{s}$, $\text{FT3D}_\text{o}$ and $\text{FT3D}_\text{p}$ to check what values the OT parameters $\epsilon, \lambda$ reach after training, to study the effect of $K$ on the FLOT's performance and compare it with that of $\text{FLOT}_0$. $\text{FT3D}_\text{p}$ is exactly the same dataset as $\text{FT3D}_\text{s}$ except that we enforce $\boldsymbol{p} + \boldsymbol{f} = \mathsf{P}\boldsymbol{q}$ when sampling the point to simulate the perfect world setting. The sole role of this ideal dataset is to confirm that the OT model holds in the perfect world, the starting point of our design.

For these experiments, training is done at $n = 2048$ for 40 epochs and takes about 9 hours. Each model is trained 3 times starting from a different random draw of $\theta$ to take into account variations due to initialisation. Evaluation is performed at $n = 2048$ on the validation sets. Note that the $n$ points are drawn at random also at validation time. To take into account this variability, validation is performed 5 different times with different draws of the points for each of the

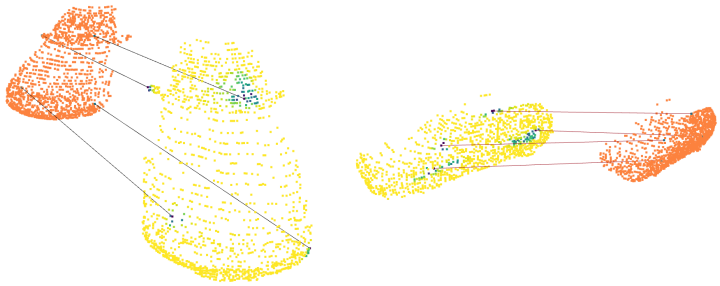

**Fig. 2.** Illustration of correspondences, found by FLOT ($K = 1$) trained on $n = 8192$ (see Section 4.4), between $\boldsymbol{p}$ and $\boldsymbol{q}$ in two different scenes of KITTI$_\mathrm{s}$. We isolated one car in each of the scenes for better visualisation. The point cloud $\boldsymbol{p}$ captured at time $t$ is represented in orange. The lines show the correspondence between a query point $\boldsymbol{p}_i$ and the corresponding point $\boldsymbol{q}_{j^*}$ in $\boldsymbol{q}$ on which most the mass is transported: $j^* = \mathrm{argmax}_j\, \mathsf{T}_{ij}$. The colormap on $\boldsymbol{q}$ represents the values in $\mathsf{T}_i$ where yellow corresponds to 0 and blue indicates the maximum entry in $\mathsf{T}_i$ and show how the mass is concentrated around $\boldsymbol{q}_{j^*}$.

trained model. For each score and model, we thus have access to 15 values whose mean and standard deviation are reported in Table 1. We present the scores obtained before and after refinement by $h$.

First, we notice that $\epsilon = 0.03$ for all model after training. We recall that we applied a constant offset of 0.03 to prevent numerical errors occurring in Alg. 1 in the exponential function when reaching to small value of $\epsilon$. Hence, the entropic regularisation, or, equivalently, the temperature in FLOT$_0$, reaches its smallest possible value. Such small values favour sparse transport plans $\mathsf{T}$, yielding sparse correspondences between $\boldsymbol{p}$ and $\boldsymbol{q}$. An illustration of these sparse correspondences is provided in Fig. 2. We observe that the correspondences are accurate and that the mass is well concentrated around the target points, especially when these points are near corners of the object.

Second, the power $\lambda/(\lambda + \epsilon)$, which controls the mass regularisation, reaches higher values on FT3D$_\mathrm{p}$ than FT3D$_\mathrm{o}$. This is the expected behaviour as FT3D$_\mathrm{p}$ contains no imperfection and FT3D$_\mathrm{o}$ contains occlusions. The values reached on FT3D$_\mathrm{s}$ are in between those reached on FT3D$_\mathrm{p}$ than FT3D$_\mathrm{o}$. This is also the expected behaviour as FT3D$_\mathrm{s}$ is free of occlusions and the only imperfections are the different sampling of the scene as $t$ and $t + 1$.

Third, on FT3D$_\mathrm{p}$, FLOT reduces by 2 the EPE compared to FLOT$_0$, which nevertheless already yields good results. Increasing $K$ from 1 to 3 further reduces the error and stabilises at $K = 5$. This validates the OT model in our the perfect world setting: the OT optimum and perfect world optimum coincide.

Fourth, on FT3D$_\mathrm{s}$ and FT3D$_\mathrm{o}$, the average scores are better for FLOT than FLOT$_0$, except for two metrics at $K = 5$ on FT3D$_\mathrm{s}$. The nevertheless good performance of FLOT$_0$ indicates that most of it is due to the trained transport

**Table 2.** Performance on $FT3D_s$ and $KITTI_s$. The scores of FlowNet3D and HPLFlowNet are obtained from [11]. We also report the scores of PointPWC-Net available in [46], as well as those obtained using the official implementation†. Italic entries are for methods publicly available but not yet published at submission time.

| Dataset | Method | EPE | AS | AR | Out. | Size (MB) |
|---|---|---|---|---|---|---|
| $FT3D_s$ | FlowNet3D [15] | 0.114 | 41.2 | 77.1 | 60.2 | 15 |
| | HPLFlowNet [11] | 0.080 | 61.4 | 85.5 | 42.9 | 77 |
| | FLOT ($K = 1$) | **0.052** | **73.2** | **92.7** | **35.7** | **0.44** |
| | *PointPWC-Net [46]* | *0.059* | *73.8* | *92.8* | *34.2* | *30* |
| | *PointPWC-Net†* | *0.055* | *79.0* | *94.4* | *29.8* | *30* |
| $KITTI_s$ | FlowNet3D [15] | 0.177 | 37.4 | 66.8 | 52.7 | 15 |
| | HPLFlowNet [11] | 0.117 | 47.8 | 77.8 | 41.0 | 77 |
| | FLOT ($K = 1$) | **0.056** | **75.5** | **90.8** | **24.2** | **0.44** |
| | *PointPWC-Net [46]* | *0.069* | *72.8* | *88.8* | *26.5* | *30* |
| | *PointPWC-Net†* | *0.067* | *78.5* | *90.6* | *22.8* | *30* |

cost $\mathsf{C}$. On $FT3D_s$ and $FT3D_o$, changing $K$ from 1 to 3 has less impact on the EPE than on $FT3D_p$. We also detect a slight decrease of performance when increasing $K$ from 3 to 5. The OT model (2) can only be an approximate model of the (simulated) real-world. The real-world optimum and OT optimum do not coincide. Increasing $K$ brings us closer to the OT optimum but not necessarily always closer to the real-world optimum. $K$ becomes an hyper-parameter that should be adjusted. In the following experiments, we use $K = 1$ or $K = 3$.

Finally, the absence of $h$ has no effect on the performance on $FT3D_p$, with FLOT still performing better than $FLOT_0$. This shows that OT module is able to estimate accurately the ideal permutation matrix $\mathsf{P}$ on its own and that the residual network $h$ is not needed in this ideal setting. However, $h$ plays a important role on the more realistic datasets $FT3D_s$ and $FT3D_o$, with an EPE divided by around 2 when present.

### 4.4 Performance on $FT3D_s$ and $KITTI_s$

We compare the performance achieved by FLOT and the alternative methods on $FT3D_s$ and $KITTI_s$ in Table 2. We train FLOT using $n = 8192$ points, as in [11], [46]. The learning rate is set to 0.001 for 50 epochs before dividing it by 10 and continue training for 10 more epochs.

The scores of FlowNet3D and HPLFlowNet are obtained directly from [11]. We report the scores of PointPWC-net available in [46], as well as the better scores we obtained using the associated code and pretrained model.[6] The model sizes are obtained from the supplementary material of [15] for FlowNet3D, and from the pretrained models provided by [11] and [46]. HPLFlowNet, PointPWC-Net and FLOT contain 19 M, 7.7 M, and 0.11 M parameters, respectively.

[6] Code and pretrained model available at `https://github.com/DylanWusee/PointPWC`.

**Table 3.** Performance on $\text{FT3D}_\text{o}$ and $\text{KITTI}_\text{o}$.

| Dataset | Method | EPE | AS | AR | Out. |
|---|---|---|---|---|---|
| $\text{FT3D}_\text{o}$ | FlowNet3D [15] | 0.160 | 25.4 | 58.5 | 78.9 |
| | $\text{FLOT}_0$ | 0.160 | 33.8 | 63.8 | 70.5 |
| | FLOT ($K = 1$) | **0.156** | **34.3** | **64.3** | **70.0** |
| | FLOT ($K = 3$) | 0.161 | 32.3 | 62.7 | 71.7 |
| $\text{KITTI}_\text{o}$ | FlowNet3D [15] | 0.173 | 27.6 | 60.9 | 64.9 |
| | $\text{FLOT}_0$ | **0.106** | **45.3** | 73.7 | 46.7 |
| | FLOT ($K = 1$) | 0.110 | 41.9 | 72.1 | 48.6 |
| | FLOT ($K = 3$) | 0.107 | 45.1 | **74.0** | **46.3** |

FLOT performs better than FlowNet3D and HPLFlowNet on both $\text{FT3D}_\text{s}$ and $\text{KITTI}_\text{s}$. FLOT achieves a slightly better EPE than PointPWC-Net on $\text{KITTI}_\text{s}$ and a similar one on $\text{FT3D}_\text{s}$. However, PointPWC-Net achieves better accuracy and has less outliers. FLOT is the method that uses the less trainable parameters (69 times less than PointPWC-Net).

We illustrate in Fig. 3 the quality of the scene flow estimation for two scenes of $\text{KITTI}_\text{s}$. We notice that FLOT aligns correctly all the objects. We also remark that the flow $\tilde{f}$ estimated at the output of the OT module is already of good quality, even though the performance scores are improved after refinement.

### 4.5 Performance on $\text{FT3D}_\text{o}$ and $\text{KITTI}_\text{o}$

We present another comparison between FlowNet3D and FLOT using $\text{FT3D}_\text{o}$ and $\text{KITTI}_\text{o}$, originally used in [15]. We train FlowNet3D using the associated official implementation. We train FLOT and $\text{FLOT}_0$ on $n = 2048$ points using a learning rate of 0.001 for 340 epochs before dividing it by 10 and continue training for 60 more epochs.

The performance of both methods is reported in Table 3. We notice that FLOT and $\text{FLOT}_0$ achieve a better accuracy than FlowNet3D with an improvement of AS of 8.8 points on $\text{FT3D}_\text{o}$ and 17.7 on $\text{KITTI}_\text{o}$. The numbers of outliers are reduced by the same amount. FLOT at $K = 1$ performs the best with $\text{FLOT}_0$ close behind. On $\text{KITTI}_\text{o}$, the best performing model are those of $\text{FLOT}_0$ and FLOT at $K = 3$.

The reader can remark that the results of FlowNet3D are similar to those reported in [15] but worse on $\text{KITTI}_\text{o}$. The evaluation on $\text{KITTI}_\text{o}$ is done differently in [15]: the scene is divided into chunks and the scene flow is estimated within each chunk before a global aggregation. In the present work, we keep the evaluation method consistent with that of Section 4.4 by following the same procedure as in [11], [46]: the trained model is evaluated by processing the full scene in one pass using $n$ random points from the scene.

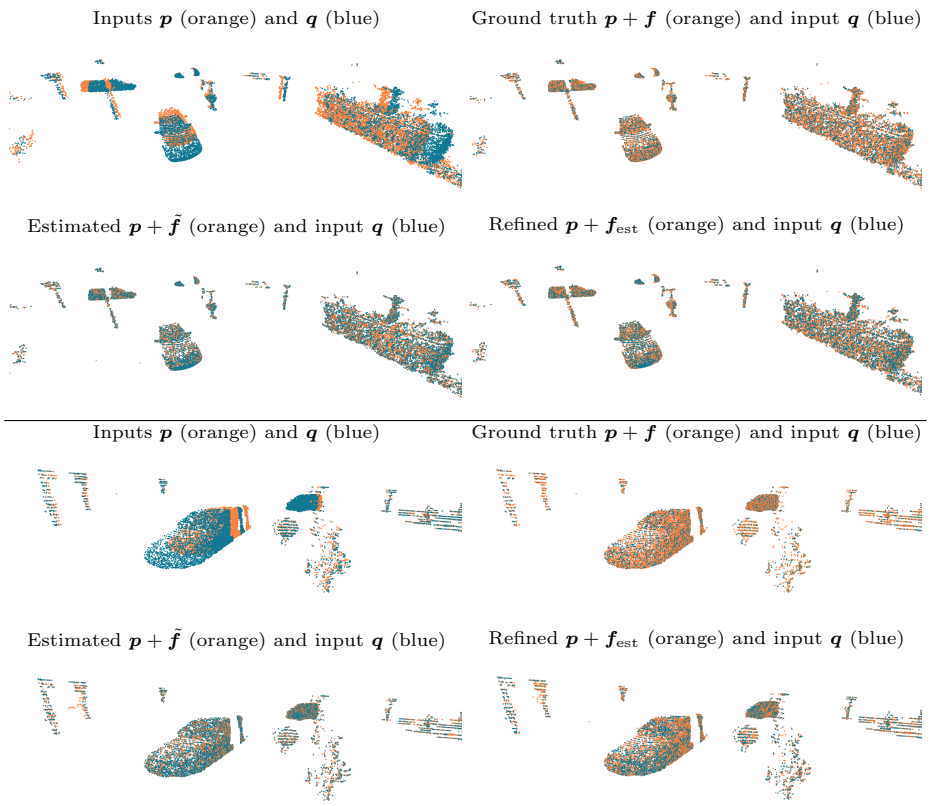


**Fig. 3.** Two scene from KITTI$_{\text{s}}$ with input point clouds $\boldsymbol{p}$, $\boldsymbol{q}$ along with the ground truth $\boldsymbol{p}+\boldsymbol{f}$, estimated $\boldsymbol{p}+\tilde{\boldsymbol{f}}$ and refined $\boldsymbol{p}+\boldsymbol{f}_{\text{est}}$ using FLOT ($K=1$) at $n=8192$.

## 5 Conclusion

We proposed and studied a method for scene flow estimation built using optimal transport tools. It can achieves similar performance to that of the best performing method while requiring much less parameters. We also showed that the learned transport cost is responsible for most of the performance. This yields a simpler method FLOT$_0$, which performs nearly as well as FLOT.

We also noticed that the presence of occlusions affects the performance of FLOT negatively. The proposed relaxation of the mass constraints in Eq. (2) permits us to limit the impact of these occlusions on the performance but does not handle them explicitly. There is thus room for improvements by detecting, *e.g.*, by analysing the effective transported mass, and treating occlusions explicitly.

**Table 4.** Architecture of $g$ and $h$ where layer $4^{(*)}$ is linear and used in $h$ only.

| Layer $\ell$ | 1 | 2 | 3 | $4^{(*)}$ |
|---|---|---|---|---|
| MLP size | 32 - 32 - 32 | 64 - 64 - 64 | 128 - 128 - 128 | 3 |

## A Networks architecture

The convolutions used in $g$ and $h$ are based on PointNet++ [30] in our implementation. Each convolution layer takes as inputs the point cloud $\boldsymbol{r} \in \mathbb{R}^{n \times 3}$ on which the convolution are performed and the features $\boldsymbol{\phi}_i^{(\ell)} \in \mathbb{R}^{c'}$, $i = 1, \ldots, n$, coming from the previous layer $\ell$. Note that these features are simply the point coordinates $\boldsymbol{r}$ at the input of $g$ and the estimated flow $\tilde{\boldsymbol{f}}$ at the input of $h$. For each point $\boldsymbol{r}_i$, the indices $\mathcal{N}(\boldsymbol{r}_i)$ of the $m = 32$ nearest neighbors to $\boldsymbol{r}_i$ in $\boldsymbol{r}$ are then computed to obtain $m$ features at point $\boldsymbol{r}_i$, each one satisfying

$$\left(\boldsymbol{\phi}_j^{(\ell)\intercal},\ \boldsymbol{r}_j^\intercal - \boldsymbol{r}_i^\intercal\right)^\intercal \in \mathbb{R}^{c'+3}, \tag{10}$$

$j \in \mathcal{N}(\boldsymbol{r}_i)$. These features are passed through a MLP : $\mathbb{R}^{c'+3} \to \mathbb{R}^{c''}$ consisting of a series of fully connected layer, instance normalisation layer with affine correction [**?**], and leaky ReLu with a negative slope of 0.1, repeated 3 times in the same order. Finally, the new feature at point $\boldsymbol{r}_i$ is obtained after passing through a final max pooling layer:

$$\boldsymbol{\phi}_i^{(\ell+1)} = \max_{j \in \mathcal{N}(\boldsymbol{p_i})} \left\{ \mathrm{MLP}\left[ (\boldsymbol{\phi}_j^{(\ell)\intercal},\ \boldsymbol{r}_j^\intercal - \boldsymbol{r}_i^\intercal)^\intercal \right] \right\} \in \mathbb{R}^{c''}, \tag{11}$$

where the max is computed independently for each of the $c''$ channels. These computations are repeated for each point $\boldsymbol{r}_i$ of the point cloud using the same MLP. The networks $g$ and $h$ share the same architecture, which is given in Table 4. Note nevertheless that the weights are not shared between $g$ and $h$.

## B Datasets

The datasets $\mathrm{FT3D_s}$ and $\mathrm{KITTI_s}$ are prepared[7] as in [11]. No occluded point remains in the processed point clouds: one can always find a point $\boldsymbol{q}_j$ in $\boldsymbol{q}$ such that $\boldsymbol{q}_j = \boldsymbol{p}_i + \boldsymbol{f}_i$ at full sampling rate $N$. However, in practice, most of the points $\boldsymbol{p}_i$ do not have a direct matching in $\boldsymbol{q}$ as both point clouds are randomly and independently sub-sampled to keep only $n \ll N$ points. This simulates different sampling of the scene. Nevertheless, no object appears or disappears because of occlusions between $t$ and $t+1$. $\mathrm{FT3D_s}$ contains $19,640$ training examples, from which we keep $2,000$ aside for validation, and $3,824$ test examples. $\mathrm{KITTI_s}$ contains 200 examples for which 142 are used for test, as in [11]. We do not use

[7] Code available at `https://github.com/laoreja/HPLFlowNet`.

the remaining KITTI examples. The ground points in $\text{KITTI}_\text{s}$ are removed using a threshold on the height. All points whose depth is larger than 35 m are removed in both datasets.

The datasets $\text{FT3D}_\text{o}$ and $\text{KITTI}_\text{o}$ are the prepared[8] by [15]. In $\text{FT3D}_\text{o}$, masks where the flow is non valid, *e.g.*, due to occlusions, are provided in used in the training loss, like in [15]. These masks are also used to compute the scores only on valid points at test time for all methods. However, the points where the corresponding flow is non-valid are present at the input of all networks. No mask is provided for $\text{KITTI}_\text{o}$. $\text{FT3D}_\text{o}$ contains $19,999$ training examples, from which we keep $2,000$ aside for validation, and $2,003$ test examples.[9] $\text{KITTI}_\text{o}$ contains 150 test examples. The ground points in $\text{KITTI}_\text{o}$ are removed by [15]. All points whose depth is larger than 35 m are removed in both datasets.

## C Performance metrics

We use the following four metrics adopted in [11], [15], [46]:

- $\text{EPE}_i = \|(\boldsymbol{f}_\text{est})_i - \boldsymbol{f}_i\|_2$: end point error, averaged over all $i$;
- AS: percentage of points such that $\text{EPE}_i < 0.05$ or $\text{EPE}_i / \|\boldsymbol{f}_i\|_2 < 0.05$;
- AR: percentage of points such that $\text{EPE}_i < 0.1$ or $\text{EPE}_i / \|\boldsymbol{f}_i\|_2 < 0.1$
- Out.: percentage of points such that $\text{EPE}_i > 0.3$ or $\text{EPE}_i / \|\boldsymbol{f}_i\|_2 > 0.1$.

The above metrics are computed as follows. The point clouds $\boldsymbol{p}, \boldsymbol{q}$ are obtained by selecting $n$ random points out of the $N$ provided points in the datasets. The flow is estimated and compared to the ground truth flow $\boldsymbol{f}$ on these $n$ selected points. The scores are averaged over the whole validation/test set.

## D Additional experimental results

### D.1 Study of FLOT

We report in Table 5 the performance of FLOT obtained at the output of the OT module on $\text{FT3D}_\text{o}$. The corresponding performance with refinement are available in the core of the paper. As on $\text{FT3D}_\text{s}$, we remark that the refinement permits to improve the EPE by around 2, confirming its utility in presence of occlusions.

### D.2 Computation time in the OT module

At $n = 2048$, the computation time[10] in the OT module is 1.4, 2.0 and 2.2 ms for $\text{FLOT}_0$, FLOT $K = 1$, FLOT $K = 3$, respectively. At $n = 8192$, the computation

[8] Datasets available at `https://github.com/xingyul/flownet3d`.

[9] We removed 8 examples with all points marked as occluded (7 in the training set and 4 in the test set). One example which contains a non valid value in the training dataset is also removed.

[10] Computed on a Nvidia GeForce RTX 2080 Ti.

**Table 5.** Performance of FLOT measured at the output of the OT module, *i.e.*, before refinement by $h$, on $\text{FT3D}_\text{o}$. We report the average scores and their standard deviations between parentheses.

| Dataset | K | EPE | AS | AR | Out. |
|---|---|---|---|---|---|
| $\text{FT3D}_\text{o}$ | $\text{FLOT}_0$ | 0.3539 (0.0028) | 6.98 (0.11) | 22.05 (0.28) | 88.76 (0.14) |
| | 1 | 0.3412 (0.0042) | 7.55 (0.17) | 23.50 (0.40) | 88.02 (0.22) |
| | 3 | 0.3426 (0.0028) | 7.38 (0.04) | 23.09 (0.05) | 88.21 (0.03) |
| | 5 | 0.3440 (0.0021) | 7.32 (0.05) | 22.94 (0.16) | 88.34 (0.09) |

time in the OT module is 13.1, 16.0, 17.9 ms for $\text{FLOT}_0$, FLOT $K = 1$, FLOT $K = 3$, respectively. This represents at most 8% of the total computation time which is itself at most of 27.8 ms at $n = 2048$ and 346 ms at $n = 8192$. Most of the time, at least 67% at $n = 2048$ and 86% at $n = 8192$, is spent in the feature extractor $g$. This shows that the OT module is responsible for just a small fraction of the total computation time.

Note that the time spent in the OT module is independent of the type of convolution used. Replacing our implementation of PointNet++ with a faster one or choosing a faster convolution will directly improve the computation time spend in $g$ and $h$. Our implementation of the OT module can also be made faster by avoiding to compute densely the cost matrix $\mathsf{C}$ by restricting the computation to points that are less than $d_{\max}$ meters apart, as these points never contribute to $\mathsf{T}$.

## References


1. Basha, T., Moses, Y., Kiryati, N.: Multi-view Scene Flow Estimation: A View Centered Variational Approach. In: Conference on Computer Vision and Pattern Recognition. pp. 1506–1513. IEEE (2010)
2. Battrawy, R., Schuster, R., Wasenmller, O., Rao, Q., Stricker1, D.: LiDAR-Flow: Dense Scene Flow Estimation from Sparse LiDAR and Stereo Images. In: International Conference on Intelligent Robots and Systems. pp. 7762–7769. IEEE (2019)
3. Baur, S.A., Moosmann, F., Wirges, S., Rist, C.B.: Real-time 3D LiDAR Flow for Autonomous Vehicles. In: Intelligent Vehicles Symposium. pp. 1288–1295. IEEE (2019)
4. Behl, A., Paschalidou, D., Donné, S., Geiger, A.: PointFlowNet: Learning representations for rigid motion estimation from point clouds. In: Conference on Computer Vision and Pattern Recognition. pp. 7962–7971. IEEE (2019)
5. Chen, Y., Pock, T.: Trainable Nonlinear Reaction Diffusion: A Flexible Framework for Fast and Effective Image Restoration. Transactions on Pattern Analysis and Machine Intelligence **39**(6), 1256–1272 (2017)
6. Chizat, L., Peyré, G., Schmitzer, B., Vialard, F.X.: Scaling algorithms for unbalanced transport problems. Mathematics of Computation **87**, 2563–2609 (2018)
7. Cuturi, M.: Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q.

(eds.) Advances in Neural Information Processing Systems. pp. 2292–2300. Curran Associates, Inc. (2013)
8. Dewan, A., Caselitz, T., Tipaldi, G.D., Burgard, W.: Rigid scene flow for 3D LiDAR scans. In: International Conference on Intelligent Robots and Systems (IROS). pp. 1765–1770. IEEE (2016)
9. Genevay, A., Peyré, G., Cuturi, M.: Learning generative models with sinkhorn divergences. In: Storkey, A., Perez-Cruz, F. (eds.) International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 84, pp. 1608–1617. PMLR (2018)
10. Gregor, K., LeCun, Y.: Learning fast approximations of sparse coding. In: International Conference on Machine Learning. pp. 399–406 (2010)
11. Gu, X., Wang, Y., Wu, C., Lee, Y.J., Wang, P.: HPLFlowNet: Hierarchical Permutohedral Lattice FlowNet for Scene Flow Estimation on Large-Scale Point Clouds. In: Conference on Computer Vision and Pattern Recognition. pp. 3249–3258. IEEE (2019)
12. Hadfield, S., Bowden, R.: Kinecting the dots: Particle based scene flow from depth sensors. In: International Conference on Computer Vision. pp. 2290–2295. IEEE (2011)
13. Kingma, D.P., Adam, J.B.: Adam : A method for stochastic optimization. In: International Conference on Learning Representations. arXiv.org (2015)
14. Liu, J., Sun, Y., Eldeniz, C., Gan, W., An, H., Kamilov, U.S.: RARE: Image Reconstruction using Deep Priors Learned without Ground Truth. Journal of Selected Topics in Signal Processing (2020)
15. Liu, X., Qi, C.R., Guibas, L.J.: FlowNet3D: Learning Scene Flow in 3D Point Clouds. In: Conference on Computer Vision and Pattern Recognition. pp. 529–537. IEEE (2019)
16. Ma, W.C., Wang, S., Hu, R., Xiong, Y., Urtasun, R.: Deep Rigid Instance Scene Flow. In: Conference on Computer Vision and Pattern Recognition. pp. 3609–3617. IEEE (2019)
17. Mardani, M., Sun, Q., Donoho, D., Papyan, V., Monajemi, H., Vasanawala, S., Pauly, J.: Neural Proximal Gradient Descent for Compressive Imaging. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. pp. 9573–9583. Curran Associates, Inc. (2018)
18. Maretic, H.P., Gheche, M.E., Chierchia, G., Frossard, P.: GOT: An Optimal Transport framework for Graph comparison. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. pp. 13876–13887. Curran Associates, Inc. (2019)
19. Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., Brox, T.: A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. In: Conference on Computer Vision and Pattern Recognition. pp. 4040–4048. IEEE (2016)
20. Meinhardt, T., Moller, M., Hazirbas, C., Cremers, D.: Learning Proximal Operators: Using Denoising Networks for Regularizing Inverse Imaging Problems. In: International Conference on Computer Vision. pp. 1799–1808. IEEE (2017)
21. Mémoli, F.: Gromovwasserstein distances and the metric approach to object matching. Foundations of computational mathematics **11**(4), 417–487 (2011)
22. Menze, M., Heipke, C., Geiger, A.: Joint 3d estimation of vehicles and scene flow. In: ISPRS Workshop on Image Sequence Analysis (2015)
23. Menze, M., Heipke, C., Geiger, A.: Object scene flow. ISPRS Journal of Photogrammetry and Remote Sensing **140**, 60–76 (2018)

24. Metzler, C., Mousavi, A., , Baraniuk, R.: Learned D-AMP: Principled Neural Network based Compressive Image Recovery. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. pp. 1772–1783. Curran Associates, Inc. (2017)
25. Mittal, H., Okorn, B., Held, D.: Just Go with the Flow: Self-Supervised Scene Flow Estimation. In: Conference on Computer Vision and Pattern Recognition. IEEE (2020)
26. Mousavi, A., Baraniuk, R.G.: Learning to invert: Signal recovery via deep convolutional networks. In: International Conference on Acoustics, Speech and Signal Processing. pp. 2272–2276. IEEE (2017)
27. Nikolentzos, G., Meladianos, P., Vazirgiannis, M.: Matching node embeddings for graph similarity. In: AAAI Conference on Articial Intelligence. pp. 2429–2435 (2017)
28. Peyré, G., Cuturi, M.: Computational optimal transport: With applications to data science. Foundations and Trends in Machine Learning **11**(5-6), 355–607 (2019)
29. Peyr, G., Cuturi, M., Solomon, J.: Gromov-Wasserstein Averaging of Kernel and Distance Matrices. In: Balcan, M.F., Weinberger, K.Q. (eds.) International Conference on Machine Learning. vol. 48, pp. 2664–2672. PMLR (2016)
30. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. pp. 5099–5108. Curran Associates, Inc. (2017)
31. Sarlin, P.E., DeTone, D., Malisiewicz, T., Rabinovich, A.: SuperGlue: Learning Feature Matching with Graph Neural Networks. In: Conference on Computer Vision and Pattern Recognition. IEEE (2020)
32. Shao, L., Shah, P., Dwaracherla, V., Bohg, J.: Motion-Based Object Segmentation Based on Dense RGB-D Scene Flow. Robotics and Automation Letters **3**(4), 3797–3804 (2018)
33. Sun, D., Yang, X., Liu, M.Y., Kautz, J.: PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. In: Conference on Computer Vision and Pattern Recognition. pp. 8934–8943. IEEE (2018)
34. Titouan, V., Courty, N., Tavenard, R., Laetitia, C., Flamary, R.: Optimal Transport for structured data with application on graphs. In: Chaudhuri, K., Salakhutdinov, R. (eds.) International Conference on Machine Learning. vol. 97, pp. 6275–6284. PMLR (2019)
35. Ushani, A.K., Wolcott, R.W., Walls, J.M., Eustice, R.M.: A learning approach for real-time temporal scene flow estimation from LIDAR data. In: International Conference on Robotics and Automation. pp. 5666–5673. IEEE (2017)
36. Ushani, A.K., Eustice, R.M.: Feature Learning for Scene Flow Estimation from LIDAR. In: Billard, A., Dragan, A., Peters, J., Morimoto, J. (eds.) Conference on Robot Learning. Proceedings of Machine Learning Research, vol. 87, pp. 283–292. PMLR (2018)
37. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is All you Need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30. pp. 5998–6008. Curran Associates, Inc. (2017)
38. Vedula, S., Baker, S., Rander, P., Collins, R., Kanade, T.: Three-dimensional scene ow. In: International Conference on Computer Vision. vol. 2, p. 722729. IEEE (1999)

39. Vogel, C., Schindler, K., Roth, S.: Piecewise rigid scene flow. In: International Conference on Computer Vision. pp. 1377–1384. IEEE (2013)
40. Wang, S., Suo, S., Ma, W.C., Pokrovsky, A., Urtasun, R.: Deep parametric continuous convolutional neural networks. In: Conference on Computer Vision and Pattern Recognition. pp. 2589–2597. IEEE (2018)
41. Wang, S., Fidler, S., Urtasun, R.: Proximal Deep Structured Models. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems. pp. 865–873. Curran Associates, Inc. (2016)
42. Wang, X., Jabri, A., Efros, A.A.: Learning Correspondence From the Cycle-Consistency of Time. In: Conference on Computer Vision and Pattern Recognition. pp. 2566–2576. IEEE (2019)
43. Wang, Y., Solomon, J.M.: Deep Closest Point: Learning Representations for Point Cloud Registration. In: International Conference on Computer Vision. pp. 3522–3531. IEEE (2019)
44. Wang, Y., Solomon, J.M.: PRNet: Self-Supervised Learning for Partial-to-Partial Registration. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. pp. 8814–8826. Curran Associates, Inc. (2019)
45. Wedel, A., Rabe, C., Vaudrey, T., Brox, T., Franke, U., Cremers, D.: Efficient dense scene flow from sparse or dense stereo data. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) European Conference on Computer Vision. pp. 739–751. Springer Berlin Heidelberg (2008)
46. Wu, W., Wang, Z., Li, Z., Liu, W., Fuxin, L.: PointPWC-Net: A Coarse-to-Fine Network for Supervised and Self-Supervised Scene Flow Estimation on 3D Point Clouds. arXiv:1911.12408v1 (2019)
47. Zou, C., He, B., Zhu, M., Zhang, L., Zhang, J.: Learning motion field of LiDAR point cloud with convolutional networks. Pattern Recognition Letters **125**, 514–520 (2019)