A. Supplementary Material

A.1. Implementation Details

Our code is implemented using PyTorch for automatic differentiation. We use Adam [21] with a learning rate of 0.015 over 500 epochs to optimize the parameters. Optimizing 36 bounding boxes over a single pair of point clouds with 40,000 points each takes about 1.7 minutes and uses 1.4 GB on a Nvidia Titan RTX GPU. While this is slow, it is still sufficient for performing offline inference over a small dataset, and since the memory consumption is modest, it can be easily parallelized for faster inference over large datasets as well. This could potentially be used to annotate data for supervised learning methods. Additionally, because our method does not need to train, we do not share the exorbitant training memory requirements of most learning based methods.

In the stereo setting, the initial bounding box grid has grid cells 4 m wide and 8 m long. For LiDAR they are instead 6 m long. Every other column is shifted forward by half the length of the grid cell, forming a diamond grid pattern, as shown in Figure 5.

Some of our scene flow parameters cannot be optimized directly and need to be computed from latent parameters. Inspired by [23], we represent rotations as 3×3 matrices and project them onto $SO(3)$ via SVD. We find that confining the per-box rotations to $SE(2)$ on the ground plane produces more consistently accurate scene flow predictions in practice. The confidences are computed by applying a sigmoid function to latent logits. The latent variables for the bounding box dimensions are exponentiated and then multiplied by the default anchor box dimensions to compute the true dimensions of the bounding box. Lastly, the heading is parameterized by a 2D vector and converted to an angle via $atan2$.

When computing the NND, in practice, we concatenate point cloud normals to the xyz coordinates and compute the NND in 6 dimensions to draw better correspondences. Point cloud normals are computed using [42], which finds the main principal vector for a local region around each point. In our case, we use the 30 nearest neighbors.

To improve efficiency, when computing the differentiable bounding box weights, we only keep points with weight above $1e{-}6$. Additionally, we only compute the loss on boxes that contain points with weight above this threshold i.e. they are not empty.

In Table 5, we list the hyperparameters we use on our various datasets. Although there are several parameters to tune, they are primarily adjusted to account for the sparsity of point clouds in the dataset. Sparser point clouds require larger ϵ and λ_{mass} but smaller n_{min}. From Table 4, one can see that our accuracy is not very sensitive to changes in L_{angle}. All other terms are largely the same across datasets.

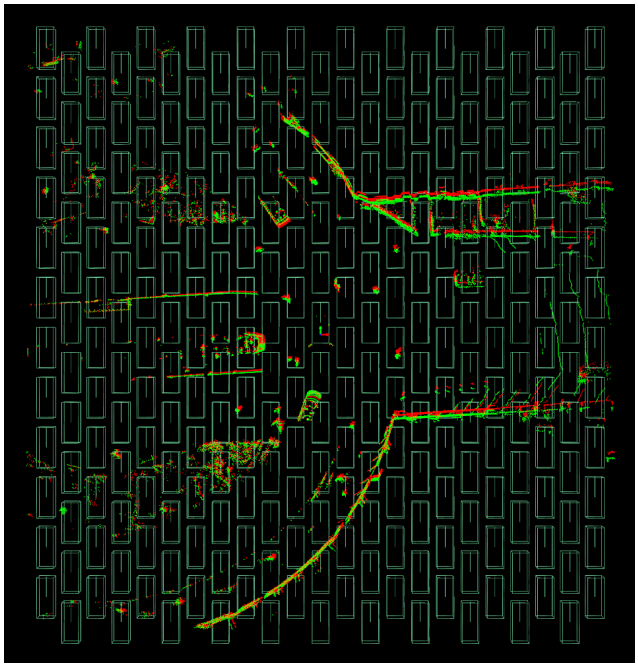


Figure 5: Initial bounding boxes during optimization.

A.2. Computing Box Coordinates

To compute the membership weights of a differentiable 3D bounding box, we need to transform the points into the local coordinate frame of the box, as mentioned in Section 3.3. Given the center of the bounding box x, y, z and the heading angle θ, the transformed points can be computed as:

$$\mathbf{p}_{box} = \mathbf{R}_{box}(\mathbf{p} - \mathbf{t}_{box}) \tag{13}$$

where $\mathbf{p}$ is the input point, $\mathbf{p}_{box}$ is the transformed point, $\mathbf{t}_{box}$ is the center of the bounding box, and $\mathbf{R}_{box}$ is the following rotation matrix parameterized by the yaw angle of the box, θ:

$$\mathbf{R}_{box} = \begin{bmatrix} -sin(\theta) & cos(\theta) & 0 \\ cos(\theta) & sin(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

A.3. Motion Segmentation Evaluation on SemanticKITTI

We also evaluate our method's segmentation accuracy on SemanticKITTI. We only consider moving vehicles and cyclists, as pedestrians move too slowly and non-rigidly for our method to detect. Initially, we found our approach struggled with the sparsity and occlusion present in SemanticKITTI, resulting in many false positives. To address this, during inference, we filter out any positive predictions that move less than 0.2 meters, corresponding to about 4.5

Table 5: Hyperparameters

Dataset	k	ϵ(m)	λ_{shape}	$\lambda_{heading}$	λ_{angle}	λ_{mass}	n_{min}
StereoKITTI	8	0.01	8	1000	0.01	0.001	500
StereoKITTI Downsampled	8	0.02	8	1000	0.01	0.002	80
LidarKITTI	8	0.03	8	1000	0.01	0.002	50
nuScenes	8	0.04	8	1000	0.25	0.01	20
SemanticKITTI	8	0.05	8	1000	0.25	0.002	50

Table 6: Dataset Details. N refers to the number of points in a point cloud.

Dataset	median N	flow	ego-motion	segmentation	correspondences
StereoKITTI	25218.5	✓	✓	✓	✓
LidarKITTI	4388.5	✓	✓	✓	
SemanticKITTI	67532.5		✓	✓	
nuScenes	6727.5	✓	✓	✓	

miles per hour, which we assume to be the minimum speed of a dynamic car or cyclist. Additionally, we introduce a cycle consistency check. Specifically, for every bounding box b_i, we optimize an additional $SE(3)$ transform T_i^i and confidence parameter c_i^i minimizing the loss function, but this time using the NND from P_2 to P_1 and computing the differentiable box weights using b_i transformed according to its forward transform T_i. During inference, we only keep boxes where both confidence scores are above the threshold. Furthermore, we assert that the two rigid transforms must be similar to each other. We measure their similarity by transforming the corners of the box by $T_i^i T_i$ and computing the average displacement of the corners, removing any predictions where the average displacement exceeds 0.2 m. After applying these two checks to reduce false positives, our method achieves 34.5% IoU on moving points, defined as:

$$IoU = \frac{TP}{TP+FP+FN} \tag{15}$$

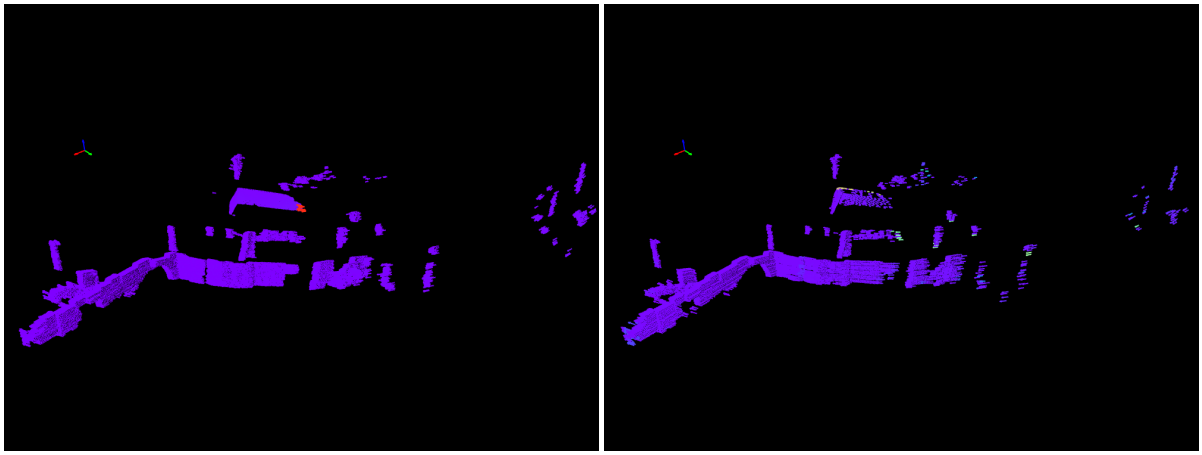
LiDAR MOS [5], a state-of-the-art supervised method, achieves 56% IoU, performing significantly better. However, it requires ground-truth segmentation masks for supervised training, while our approach does not require any labels. Note that LiDAR MOS preprocesses its point clouds differently and does not exclude pedestrians in its evaluation.

A.4. More Visualizations

We visually compare our method against another scene in KITTI (Figure 6) and two scenes from nuScenes (Figure 7). These are visualized similarly to Figure 3. Again, our method performs the best, while PointPWCNet and NSFP are noisy and struggle to accurately predict the motion of dynamic objects.

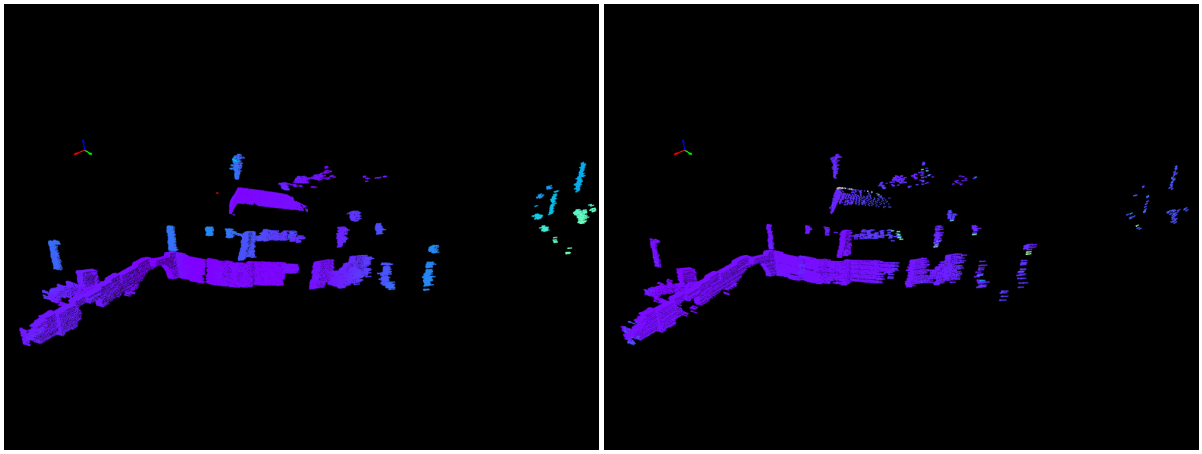
For a more exhaustive visualization of just our method, we also display our predictions over six scenes from StereoKITTI and LidarKITTI, visualized in 4 different ways, shown in Figures 8, 9, 10, 11, 12, 13. Our method is able to predict accurate scene flow and segmentation masks on all scenes using both stereo and LiDAR, although the stereo predictions are slightly more accurate. This is evidenced in that while the LiDAR predictions successfully detect most moving objects, the stereo predictions detect all of them, illustrated in Figures 8 and 12. As a note, in Scenes 11 and 13, there are moving bikers and motorcyclists. These points are cropped out in StereoKITTI, but with LidarKITTI, we utilize the entire point cloud, so our method detects these moving objects. They are shown as empty boxes in the third visualization on these scenes.

Additionally, we include animated GIFs of the bounding box optimization process for StereoKITTI in our supplementary zip file. The color of the boxes indicate their confidence score, with purple corresponding to 0 and red corresponding to 1.



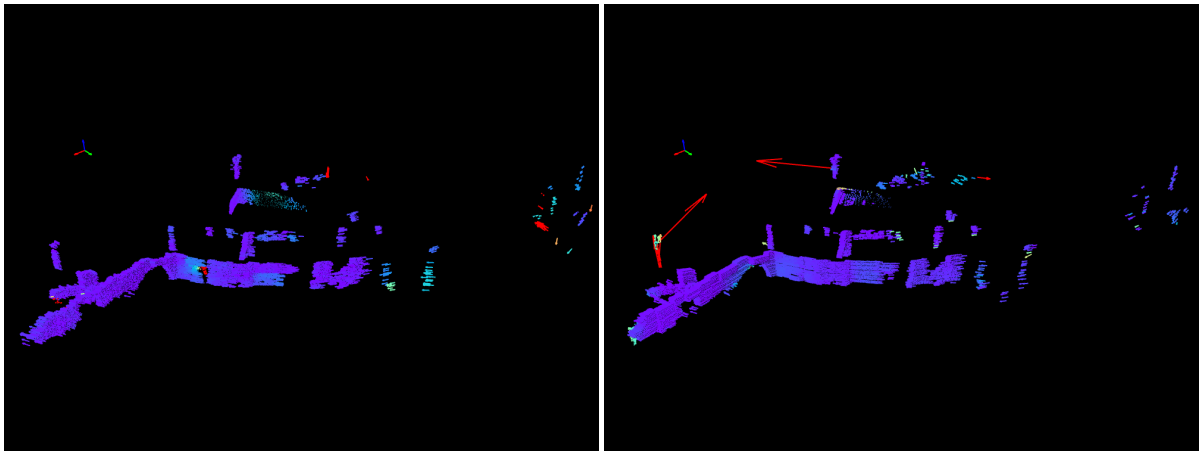
(a) Ours StereoKITTI

(b) Ours LidarKITTI



(c) NSFP StereoKITTI

(d) NSFP LidarKITTI

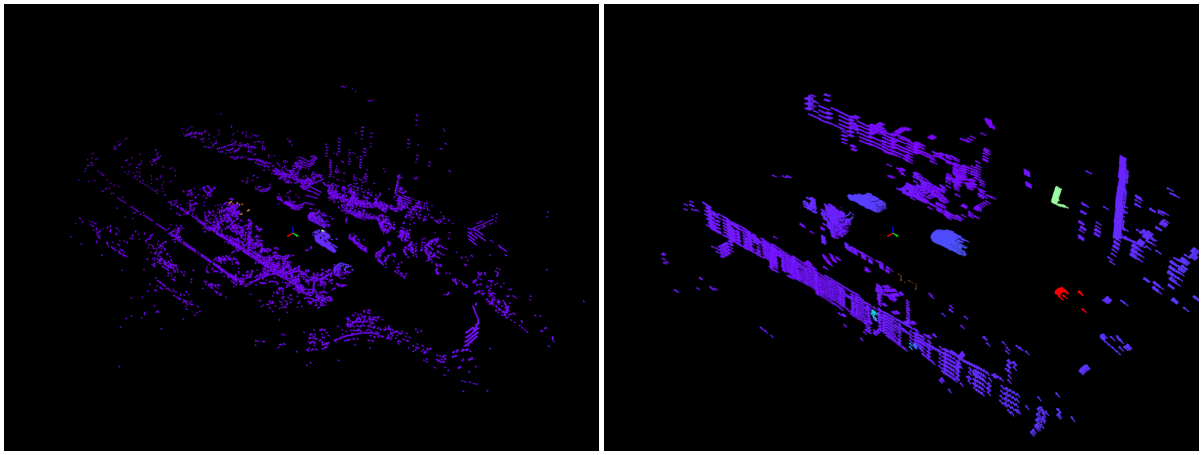


(e) PointPWCNet StereoKITTI

(f) PointPWCNet LidarKITTI

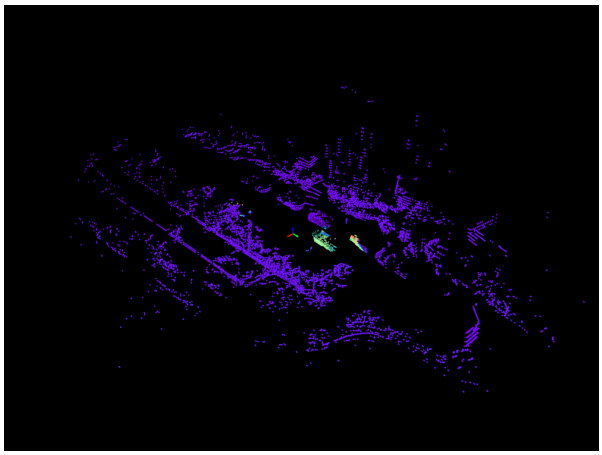
(g) Error Colorbar

Figure 6: Visualization of scene flow predictions for our method, NSFP, and the PointPWCNet loss function under direct optimization on another scene in KITTI. Color indicates the EPE3D of the prediction, with red indicating high error and purple indicating low error. For StereoKITTI, the colorscale ranges from 0-0.5 m error, while for LidarKITTI, it ranges from 0-1 m. In this scene, a van is driving ahead of the moving ego vehicle. Our method and NSFP are able to accurately predict the flow on this scene in both settings, although our method is slightly more accurate. PointPWC also generally performs well but struggles in the sparser regions of the point cloud.

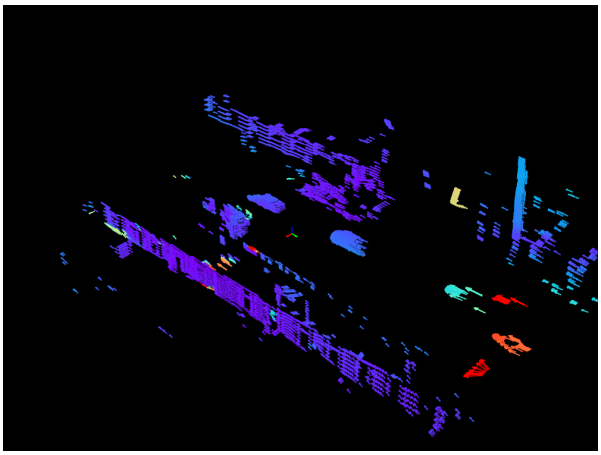


(a) Ours Scene 1

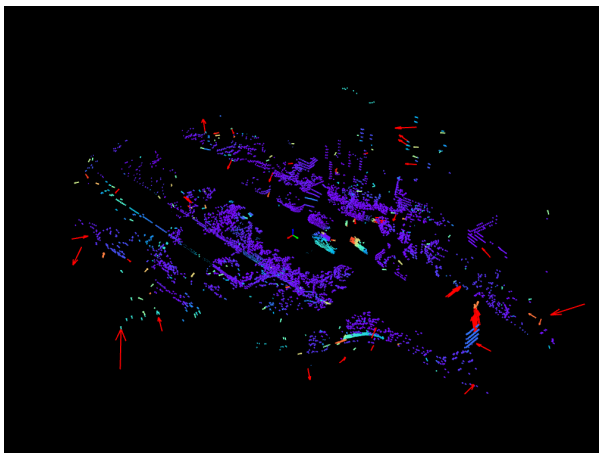
(b) Ours Scene 2



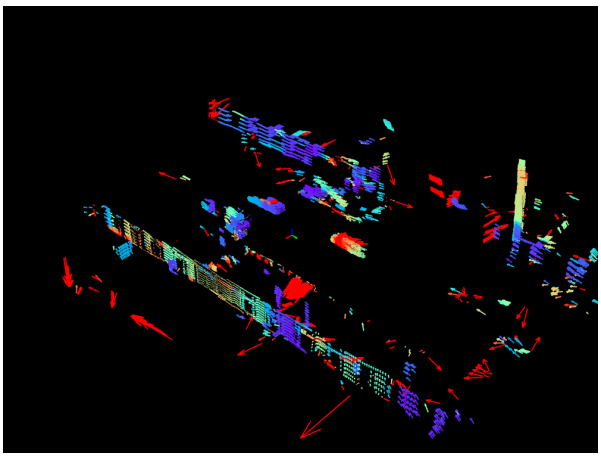
(c) NSFP Scene 1



(d) NSFP Scene 2



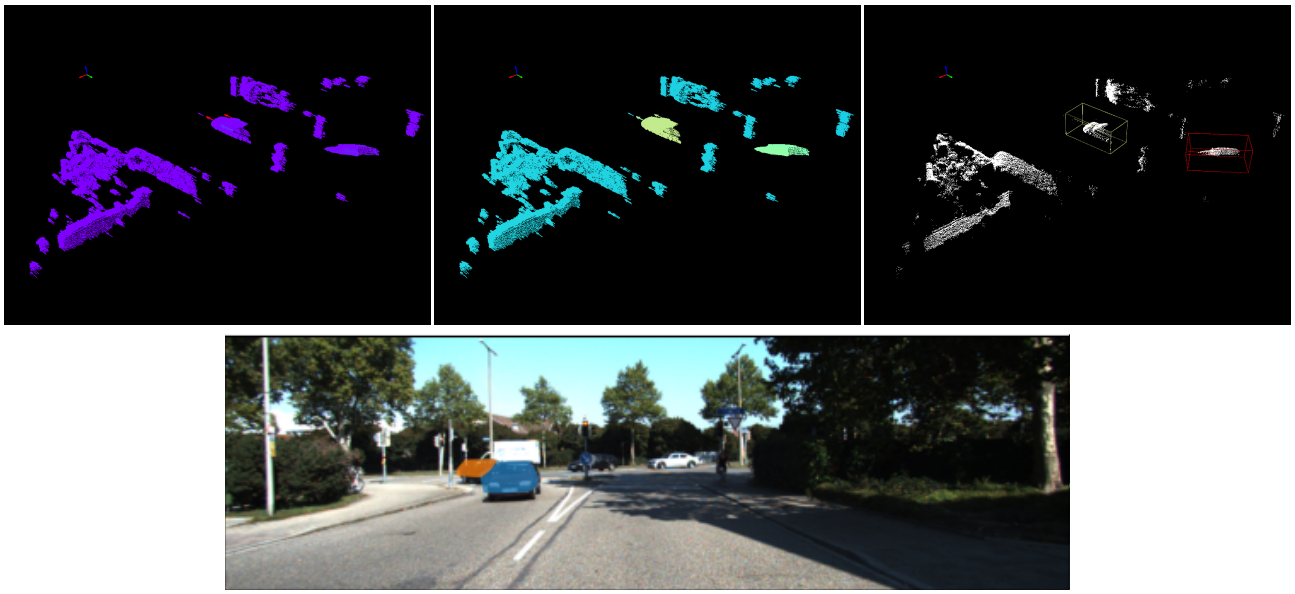
(e) PointPWCNet Scene 1



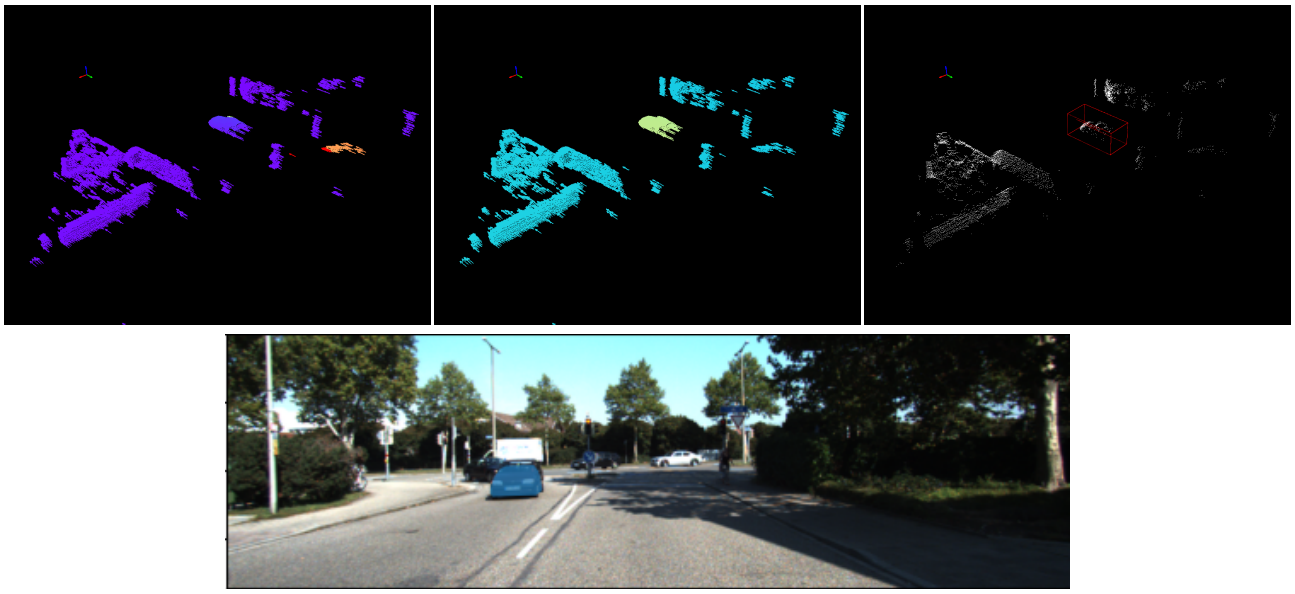
(f) PointPWCNet Scene 2

(g) Error Colorbar

Figure 7: Visualization of scene flow predictions for our method, NSFP, and the PointPWCNet loss function under direct optimization on two scenes in nuScenes. Color indicates the EPE3D of the prediction, with red indicating 1 m and purple indicating 0 m. In scene 1, the ego vehicle is driving forward along with two cars ahead of it and one behind it. Our method is able to predict the motion of all cars but the one behind, due to the sparsity of points on that car. NSFP struggles with two of the moving cars and also falsely predicts the motion of a parked car. PointPWC's prediction exhibits a lot of artifacts, especially at the boundary of the scene. In scene 2, the ego vehicle approaches an intersection as another car drives close behind. In the other lane, three cars move in the opposite direction. Another car moves along the perpendicular street of the intersection, totalling five dynamic vehicles in this scene. Our method successfully predicts three of them, while the other methods show large errors on static parts of the scene and are less accurate on the dynamic objects as well.

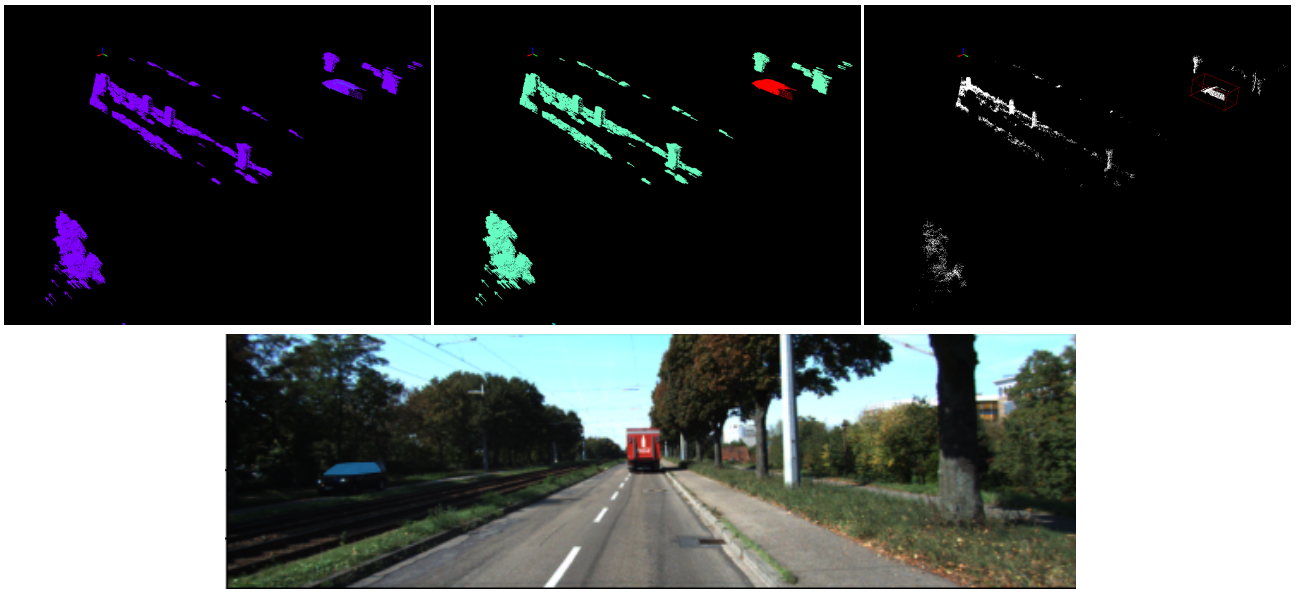


(a) StereoKITTI

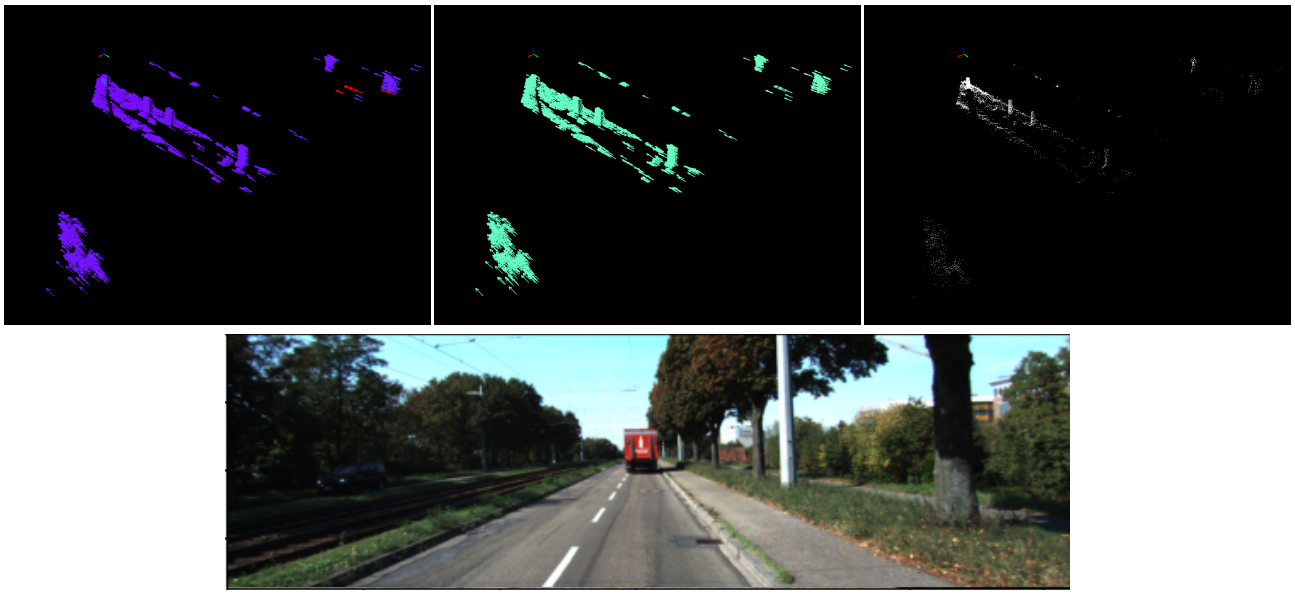


(b) LidarKITTI

Figure 8: Scene 1 Visualizations. (a) shows StereoKITTI predictions and (b) show LidarKITTI predictions on the same scene. The left visual in the first row of each subfigure shows the 3D end-point-error of our predictions, similar to Figures 3, 6, 7. The color can be interpreted using the same colorbar as the comparative visualizations, but with purple corresponding to 0 m and red corresponding to 0.75 m. The middle visual shows the magnitude of the predicted scene flow vectors using the same colorbar, with purple corresponding to 0 m and red corresponding to 2.5 m. The right visual shows the predicted bounding boxes using arbitrary colors. Lastly, the bottom visual projects a convex hull of the segmented points onto the image plane for each detected moving object, performing moving object instance segmentation on images. These colors are also arbitrary. This figure displays the same scene as 3.

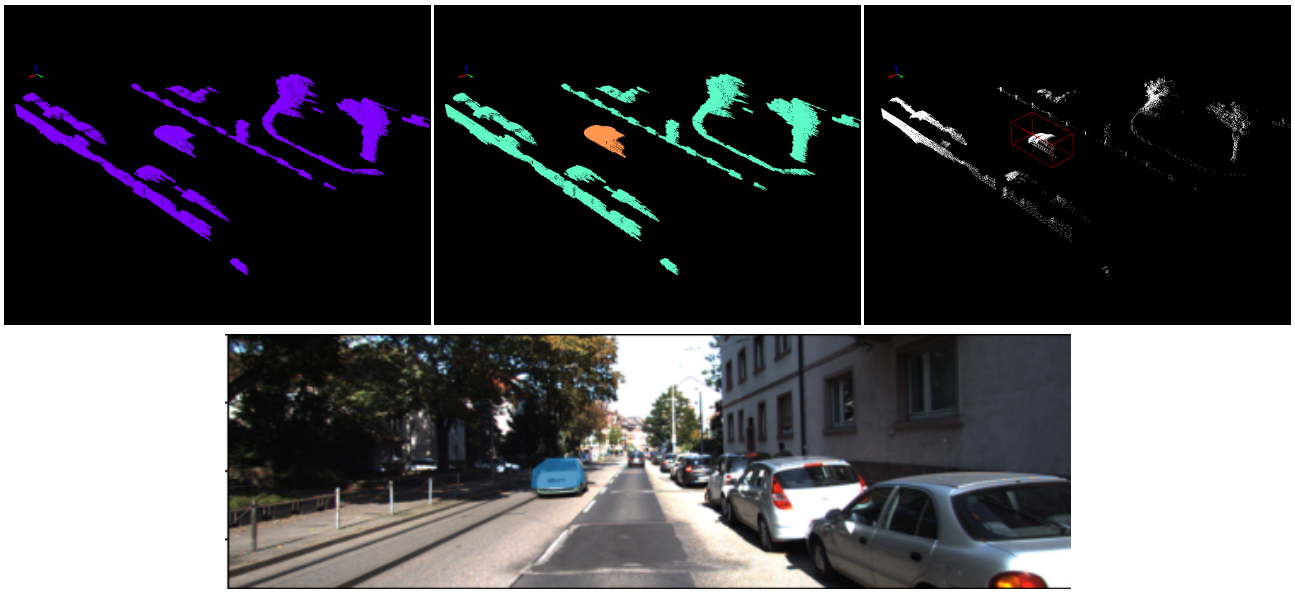


(a) StereoKITTI

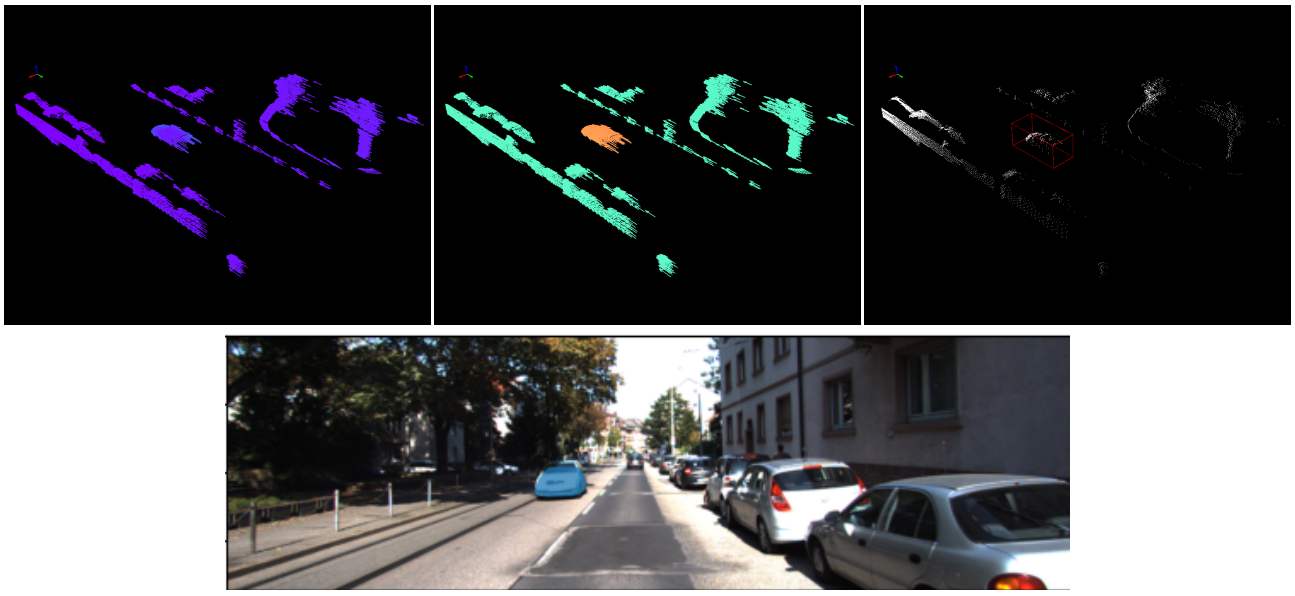


(b) LidarKITTI

Figure 9: Scene 2 Visualizations. Same as Figure 8. In this scene, the ego vehicle is moving fast on a main street as an oncoming car approaches on the other side of the road. Our method is able to identify the moving car in the stereo setting, but in the LiDAR setting, the points on the car are extremely sparse, making it difficult for our method to identify it.

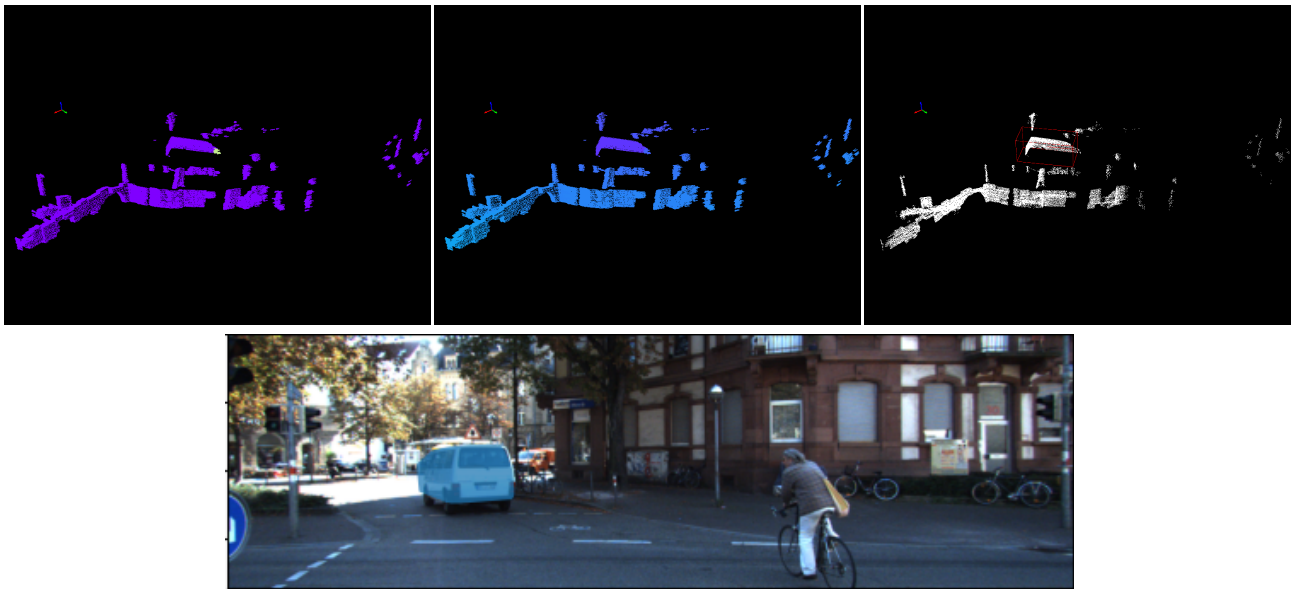


(a) StereoKITTI

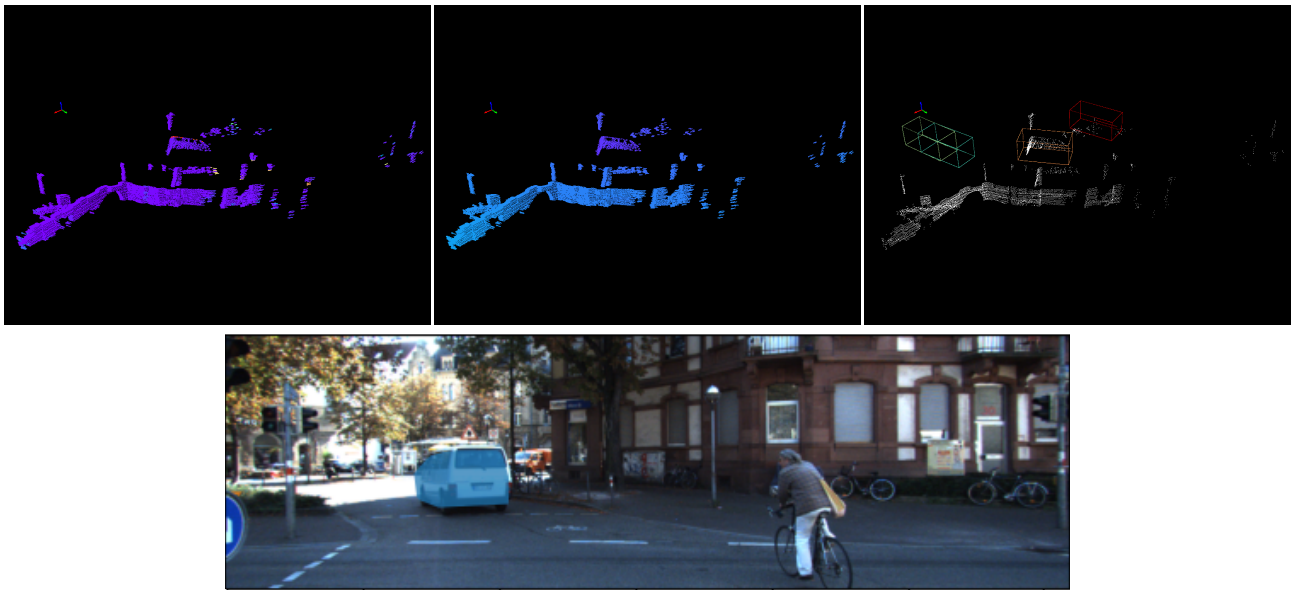


(b) LidarKITTI

Figure 10: Scene 3 Visualizations. Same as Figure 8. In this scene, the ego vehicle drives forward slowly on a narrow street as another car approaches in the opposite lane. Our method is able to accurately predict the flow on the dynamic car in both settings.

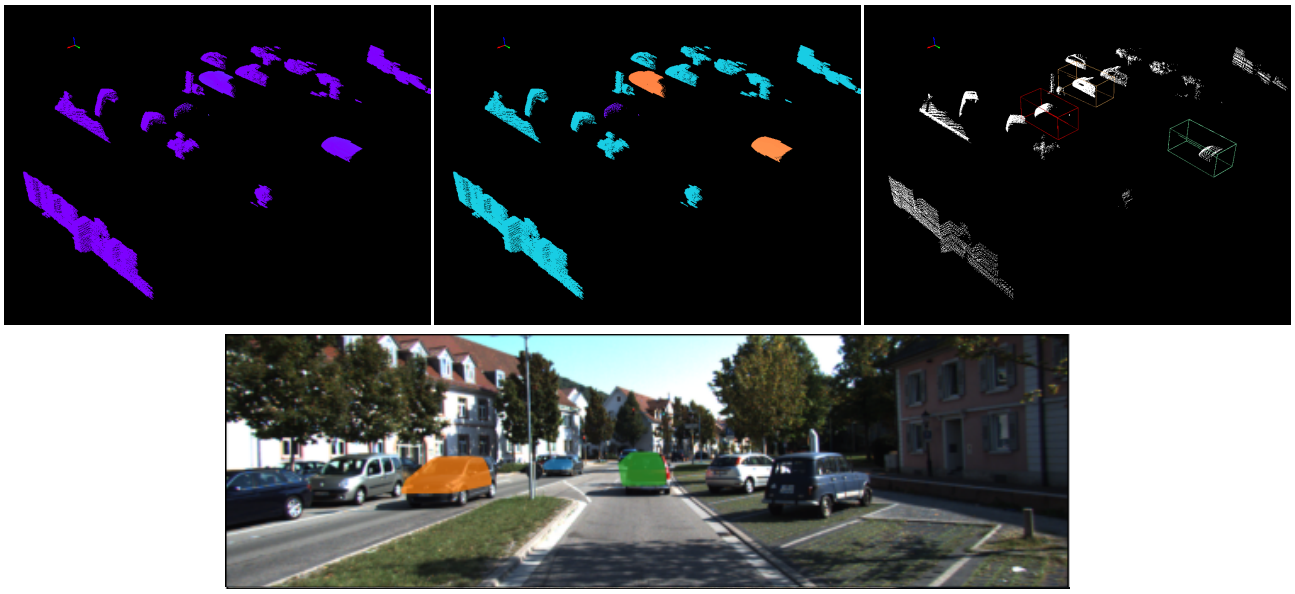


(a) StereoKITTI

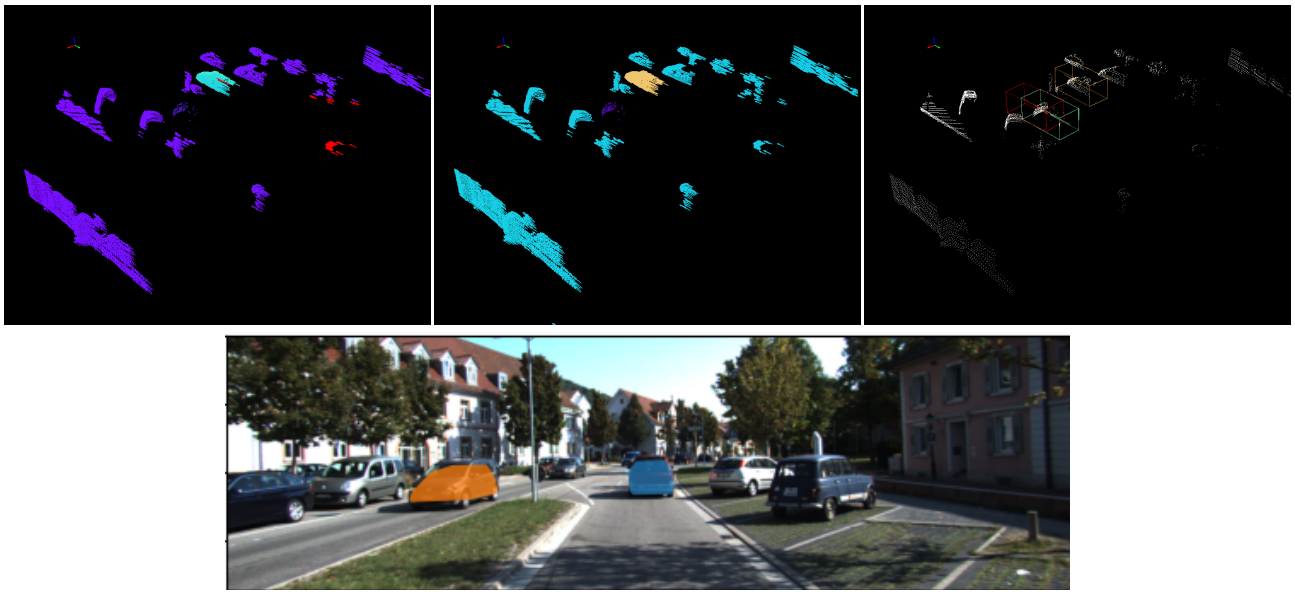


(b) LidarKITTI

Figure 11: Scene 4 Visualizations. Same as Figure 8. This figure displays the same scene as 6. Note that there is a biker in the scene. These points are cropped out in StereoKITTI as they don't posses ground truth scene flow annotations, but with LidarKITTI, we utilize the entire point cloud, so our method is able to detect the biker, as shown by the empty green box.

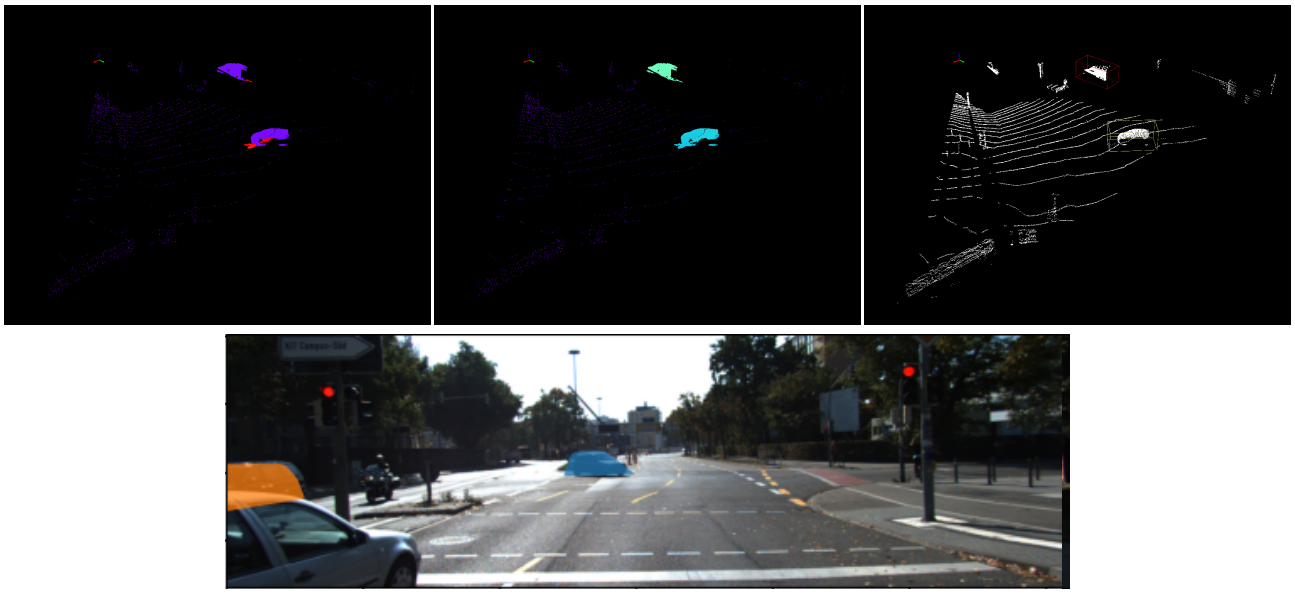


(a) StereoKITTI

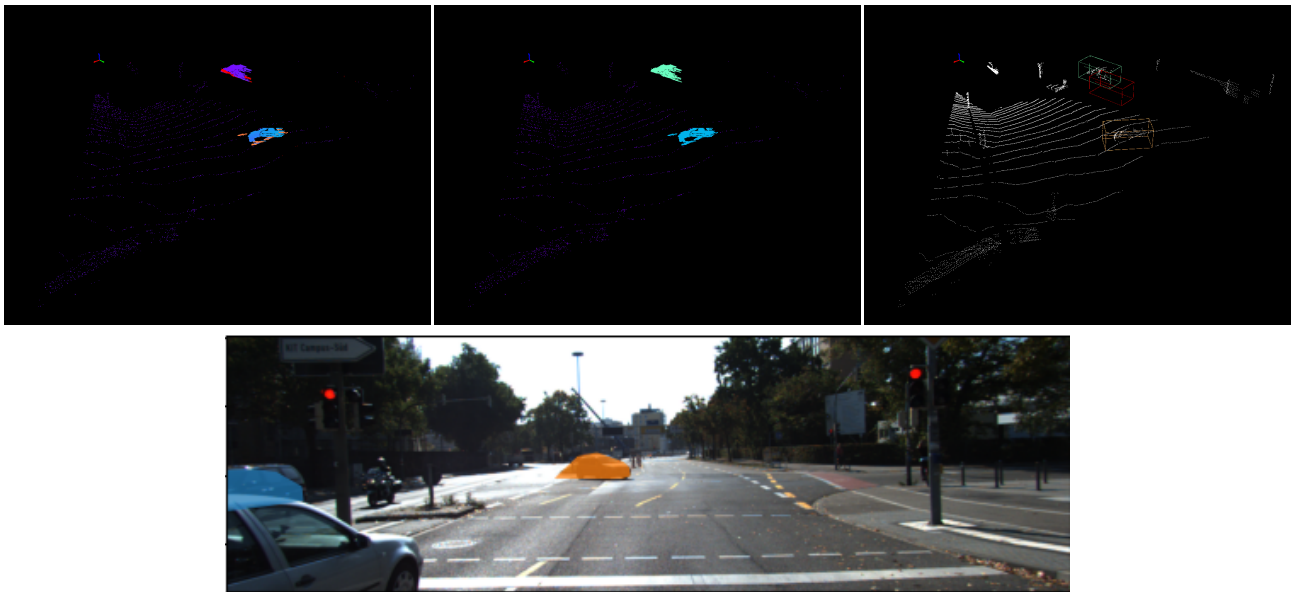


(b) LidarKITTI

Figure 12: Scene 5 Visualizations. Same as Figure 8. In this scene, the ego vehicle is driving behind another car as two cars approach from the opposite lane. Our method predicts all three moving cars in the stereo setting, but misses the furthest car in the LiDAR setting due to its sparsity.



(a) StereoKITTI



(b) LidarKITTI

Figure 13: Scene 6 Visualizations. Same as Figure 8. In this scene, the ego vehicle is stopped at a stop light at an intersection while a car and a motorcyclist cross the intersection and approach from the other side of the street. Additionally, another car coming from the same direction makes a left turn at the intersection. Our method is able to identify both moving cars in both settings. Similar to 12, the points of the motorcyclist are not present in the stereo setting but are present in the LiDAR setting, and our method successfully identifies at, as indicated by the empty red box.