R3D3: Dense 3D Reconstruction of Dynamic Scenes from Multiple Cameras Supplementary Material

Aron Schmied[1*] Tobias Fischer[1*] Martin Danelljan[1] Marc Pollefeys[1,2] Fisher Yu[1]
[1] ETH Zürich [2] Microsoft
https://www.vis.xyz/pub/r3d3/

This supplementary material provides further details on our method, our experimental setup, and more quantitative and qualitative results and comparisons.In Sec. 1, we provide further details and discussion on the geometric pose and depth estimation, the refinement network, our covisibility graph, and training and inference procedures. In Sec. 2, we provide additional ablation studies and more quantitative and qualitative comparisons and results. In Sec. 3, we provide a detailed derivation of our multi-camera DBA. Note that we group large qualitative comparisons at the end of this document for better readability.

1. Implementation Details

1.1. Geometric Depth and Pose Estimation

We follow [7] for the context feature encoder g_ψ and correlation feature encoder g_ϕ and GRU architecture. In addition to the context features, g_ψ also outputs an initialization to the GRU hidden state $\mathbf{h}_{ij}^{(0)}$ from input $\mathbf{I}_i$. The hidden state after the last iteration is pooled among all outgoing edges from node i to predict a damping factor λ_i that serves as a parameter to the multi-camera DBA, which improves convergence when the current depth estimate is inaccurate. We perform feature matching and geometric depth updates at 1/8$^{\text{th}}$ of the original image resolution. We predict upsampling masks from the pooled hidden state to upsample depth to the original resolution. The confidence maps $\mathbf{w}_i$ are linearly interpolated. Furthermore, following RAFT [6], the correlation volume is built as a 4-level pyramid, and sampled features from all layers are concatenated. Further, each GRU update is followed by a multi-camera DBA with two Gauss-Newton steps.

1.2. Depth Refinement

We show the architecture of the depth refinement network in Tab. 1. Note that we input full-resolution frames and up-sampled depth and confidence. Further, we concatenate the depth and confidence predicted at 1/8$^{\text{th}}$ scale with features after the third skip connection. We use the inverse of the input depth. Further, we obtain the refined depth prediction from the sigmoid output $\mathbf{o}_t^c \in [0,1]^{H\times W}$ via

$$\frac{1}{\mathbf{d}_t^c} = \frac{f_c}{f_{\text{norm}}} \cdot \left(\frac{1}{d_{\text{max}}} + (\frac{1}{d_{\text{min}}} - \frac{1}{d_{\text{max}}}) \cdot \mathbf{o}_t^c \right), \quad (1)$$

where f_c is the focal length, f_{norm} a constant normalization factor and d_{min} and d_{max} are pre-defined minimum and maximum depth. For experiments on the DDAD dataset we set $d_{\text{min}} = 1$, $d_{\text{max}} = 200$, $f_{\text{norm}} = 715$ aligned with the focal length of the front-facing camera, for NuScenes we choose $d_{\text{min}} = 1$, $d_{\text{max}} = 80$, $f_{\text{norm}} = 500$.

Dataset generation. To train the refinement network, we first generate a dataset of samples that contain the prior geometric depth, confidence maps, and poses with the first two stages of our system. We filter scenes with inaccurate scene scale by measuring how many reliable feature matches there are across both temporal and spatial edges with the given confidences $\mathbf{w}_{ij}$. The fewer reliable matches, the weaker the constraint on the metric scale. Furthermore, based on the generated poses, we remove static scenes. This allows us to train an absolute scale monocular depth estimation model from the raw video data.

Training details. During the training of refinement network $\mathcal{D}_\theta$, we randomly set input depth and confidence weights to zero to learn depth prediction with and without prior geometric estimates as input. We use color-jitter and random horizontal flipping as augmentations. Further, we follow a two-stage training paradigm as in [3]. After training $\mathcal{D}_\theta$ in the first stage, we remove training samples with outliers in the depth estimates of the current $\mathcal{D}_\theta$. In particular, we apply RANSAC to determine the ground plane in the front view and calculate the height of each pixel in all views. We omit training samples with $\frac{1}{H\cdot W}\sum_{u,v}[h_{u,v} < -0.5m] > \epsilon$ where $h_{u,v}$ is the height of pixel (u,v) w.r.t. the ground plane and ϵ is set to 0.005 for the front and backward-facing cameras and 0.02 for the side views. This filters frames where a significant amount of pixels are below the ground plane. In the second stage,

*Equal contribution.

we re-train the network from scratch on the filtered dataset for 20 epochs with the same settings. On NuScenes, we train our refinement network with all available camera images (12Hz) instead of only keyframes (2Hz).

1.3. Co-visibility Graph

We detail our multi-camera co-visibility graph construction described in Sec. 3.1 of the main paper in Algorithm 1. Note that `GetAdjacentNodes` returns a different adjacency pattern than the spatial edges in A, as described in Sec. 3.1 of the main paper. In particular, we leverage the forward motion assumption in order to connect two frames (i, j) if camera c_j is closer to the forward-facing camera than camera c_i. For further clarification, please refer to Fig. 3 of the main paper.

Algorithm 1 Co-visibility graph construction

Input: $\mathbf{K}, \mathbf{T}, \mathcal{G}, \{\mathbf{I}_t^c\}_{c=0}^C$

A = `ComputeStaticAdjacency`($\mathbf{K}$, $\mathbf{T}$)
N = `GetNodes`($\mathcal{G}$)
M = `AddNodes`($\mathcal{G}$, $\mathbf{I}_t^c$)
for $i \in M$ # add temporal edges
  for $j \in N$
    if `Radius`$(i, j) < r_{\text{intra}}$
      `AddEdge`($\mathcal{G}$, i, j)
    end if
  end for
end for
for $(i, j) \in A$ # add spatial edges
  `AddEdge`($\mathcal{G}$, i, j)
end for # add spatial-temporal edges
O = `GetNodesAtTime`($\mathcal{G}$, $t - r_{\text{intra}}$)
O' = `GetAdjacentNodes`($\mathcal{G}$, A, M, O)
for $(i, j) \in O'$
  `AddEdge`($\mathcal{G}$, i, j)
end for
# Remove out-of-context edges and nodes
`RemoveTemporalEdges`($\mathcal{G}, t - \Delta_{t_{\text{intra}}}$)
`RemoveSpatialTemporalEdges`($\mathcal{G}, t - \Delta_{t_{\text{inter}}}$)
`RemoveUnconnectedNodes`($\mathcal{G}$)

Dynamic alternative. We further implement a dynamic algorithm without the assumptions stated in Sec. 3.1 of the main paper. The dynamic algorithm establishes edges based on camera frustum overlap, *i.e.* it measures the intersection over union (IoU) of the camera frustums of each frame across a local time window in world space given the current camera pose estimates $\mathbf{G}$. We order frame pairs by their IoU in descending order and choose the N highest overlapping pairs as co-visible frames. These establish the temporal and spatial-temporal edges.

We found empirically that the static algorithm performs similarly to the dynamic alternative while being simpler and more efficient, so we use it in our experiments. However, for applications where the assumptions in Sec. 3.1 of the main paper do not hold, this algorithm provides a suitable alternative.

1.4. Inference Details

For both datasets, we observe that camera shutters are not well synchronized in both datasets. This poses a problem, especially at high speeds. Thus, instead of using constant camera extrinsics, we compute time-dependent relative camera extrinsics. For inference, we set $n_{\text{itr-wm}} = 16$, $n_{\text{iter1}} = 4$ and $n_{\text{iter2}} = 2$. For Nuscenes, use a different threshold $\beta = 0.8$.

2. Experiments

Evaluation metrics. We evaluate the proposed method in terms of depth accuracy and trajectory accuracy.

Depth. Given the estimated depths $\mathbf{d}_t^c$ and ground truth depth $\mathbf{d}_t^{*c}$ we use compare the following depth metrics

$$\text{Abs Rel: } \frac{1}{T \cdot C} \sum_{d,c} \frac{|\mathbf{d}_t^c - \mathbf{d}_t^{*c}|}{\mathbf{d}_t^{*c}}$$

$$\text{Sqr Rel: } \frac{1}{T \cdot C} \sum_{d,c} \frac{\|\mathbf{d}_t^c - \mathbf{d}_t^{*c}\|^2}{\mathbf{d}_t^{*c}}$$

$$\text{RMSE: } \frac{1}{T \cdot C} \sqrt{\sum_{d,c} \|\mathbf{d}_t^c - \mathbf{d}_t^{*c}\|^2}$$

$$\delta_{1.25}\text{: fraction of } d \in \mathbf{d} \text{ for which } \max\left(\frac{d}{d^*}, \frac{d^*}{d}\right) < 1.25 \tag{2}$$

For up-to-scale evaluation, we resort to the camera-wise metric scaling as described in [4] which results in scaling factor s described as

$$s = \frac{1}{C} \cdot \sum_c \frac{median(d^{*c})}{median(d^c)} \tag{3}$$

We then scale the predicted depth as $s \cdot \mathbf{d}$.

Trajectory. For trajectory evaluation we use the absolute trajectory error (ATE) score. Given an estimated trajectory $\mathbf{P}_1, ..., \mathbf{P}_T \in SE(3)$ and ground truth trajectory $\mathbf{Q}_1, ..., \mathbf{Q}_T$ the ATE is defined as

$$\begin{aligned} \mathbf{F}_i &= \mathbf{Q}_i^{-1} \mathbf{S} \mathbf{P}_i \\ ATE &= RMSE(\mathbf{F}_{1:T}) = \sqrt{\frac{1}{T} \sum_i \|trans(\mathbf{F}_i)\|^2} \end{aligned} \tag{4}$$

where $trans$ defines the translational part of $\mathbf{F}$ and $\mathbf{S}$ is identity $\mathbf{I}$ for unscaled evaluation or $s_{traj} \cdot \mathbf{I}$ where s_{traj} is determined via least squares for scaled evaluation.

Table 1. **Depth refinement network architecture.** K describes the kernel size, S the stride. ResidualBlock consists of two convolutional layers and a skip-connection as proposed in [5]. We generate output at four scales in $[0, 1]$ which is normalized by focal length of the respective camera and scaled to $[1/d_{max}, 1/d_{min}]$.

No.	Input	Layer Description	K	S	Output Size
		(#A, #B) UpBlock			
#i	(#A)	Conv2d→ ELU → Up	3	1	
#ii	(#i, #B)	Concatenate → Conv → ELU	3	1	
		Encoder			
#0		**Input**: Image + Inv. Geometric Depth + Confidence			$5 \times H \times W$
#1	(#0)	Conv → BN → ReLU	7	2	$64 \times H/2 \times W/2$
#2	(#1)	MaxPool → **Skip**	3	2	$64 \times H/4 \times W/4$
#3	(#2)	2xResidualBlock → **Skip**	3	1	$64 \times H/4 \times W/4$
#4	(#3)	2xResidualBlock → **Skip**	3	2	$128 \times H/8 \times W/8$
#5		**Input**: Inv. Geometric Depth + Confidence			$2 \times H/8 \times W/8$
#6	(#3, #5)	Concatenate			$130 \times H/8 \times W/8$
#7	(#6)	2xResidualBlock → **Skip**	3	2	$256 \times H/16 \times W/16$
#8	(#7)	2xResidualBlock	3	2	$512 \times H/32 \times W/32$
		Decoder			
#9	(#8, #7)	UpBlock	3	1	$256 \times H/16 \times W/16$
#10	(#9, #4)	UpBlock	3	1	$128 \times H/8 \times W/8$
#11	(#10)	Conv2d → Sigmoid → **Output**	3	1	$1 \times H/8 \times W/8$
#12	(#10, #3)	UpBlock	3	1	$64 \times H/4 \times W/4$
#13	(#12)	Conv2d → Sigmoid → **Output**	3	1	$1 \times H/4 \times W/4$
#14	(#12, #2)	UpBlock	3	1	$32 \times H/2 \times W/2$
#15	(#14)	Conv2d → Sigmoid → **Output**	3	1	$1 \times H/2 \times W/2$
#16	(#15)	UpBlock	3	1	$16 \times H \times W$
#17	(#16)	Conv2d → Sigmoid → **Output**	3	1	$1 \times H \times W$

Ablation studies. In Tab. 3, we show a comparison of our depth refinement network trained with synthetic data only, with both synthetic data and real-world data, and real-world data only. As stated in Sec. 3.3 of the main paper, the results show that synthetic data cannot help the depth refinement network performance, even when fine-tuning on real-world data afterward. Instead, we observe the best performance when starting training from real-world data directly. This underlines the importance of our self-supervised training scheme for the refinement network, since contrary to the geometric parts of our system, here we cannot rely on synthetic data to provide us with ground-truth supervision.

For Tab. 4, we train two versions of our refinement network described in Sec. 3.3 of the main paper. We train $\mathcal{D}_\theta$ as described in the main paper with sparsified input depth and train $\mathcal{D}_\omega$ purely for monocular depth estimation without refining geometric estimates. We evaluate both networks on monocular depth estimation on DDAD. We observe that the networks perform similarly, while $\mathcal{D}_\theta$ can both refine geometric depth estimates and estimate depth without geometric depth input. This shows that the refinement network learns strong scene priors that can estimate depth even without any additional input. Further, we can conclude that the network generalizes well to both depth prediction and depth refinement.

In Fig. 1 we show an ablation of the covisibility graph density by varying the parameters described in Sec. 3.1 of our paper. An increase in the number of edges in the covisibility graph only yields marginal improvement, while resulting in a linear increase in runtime.

Table 2. **Masking scheme ablation on DDAD.** We compare the influence of the different masks used to train the refinement network (cf. Eq. 8 of the main paper) on the final performance.

$\mathbf{M}^{\text{st}}$	$\mathbf{M}^{\text{oc}}$	$\mathbf{M}^{\text{fc}}$	Abs Rel ↓	Sq Rel ↓	RMSE ↓	$\delta_{1.25}$ ↑
			0.637	51.086	18.129	0.779
✓			0.288	10.304	12.982	0.786
✓	✓		**0.162**	3.304	11.638	**0.811**
✓	✓	✓	**0.162**	**3.019**	**11.408**	**0.811**

During training of $\mathcal{D}_\theta$, we mask regions that do not provide useful information for depth learning when minimizing the view synthesis loss in Eq. 8 of the paper. We provide an ablation study of the three masks used in Eq. 8 in Tab. 2. The self-occlusion $\mathbf{M}^{\text{oc}}$ and static $\mathbf{M}^{\text{st}}$ masks are essential to depth learning by removing ego-vehicle and *e.g.* sky regions, respectively. The flow consistency mask $\mathbf{M}^{\text{fc}}$ further reduces outliers, caused by *e.g.* dynamic objects.

Qualitative comparisons. In Fig. 3, we qualitatively compare our method to existing works in terms of scale-aware depth estimation on DDAD. Further, we show our geometric depth estimate alongside the refined depth. We notice that, especially for the side views, existing works struggle to obtain accurate depth. On the other hand, the geometric depth of our method produces many accurate depth predic-

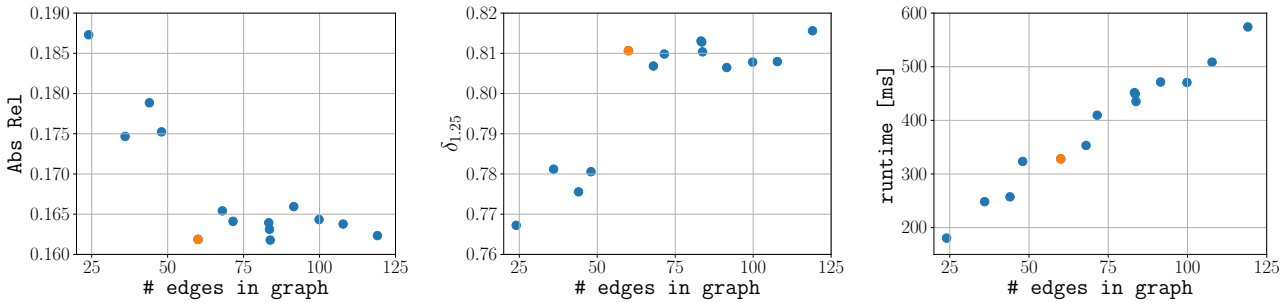

Figure 1. **Graph density analysis on DDAD.** We plot the AbsRel, $\delta_{1.25}$ scores and runtime w.r.t. the number of edges in the co-visibility graph. Our default setting is shown in orange.

Table 3. **Refinement network training.** We show that training and pre-training the refinement network on the synthetic VKITTI [1] dataset does not yield improvement over self-supervised training with real-world data as proposed in Sec. 3.3 of the main paper.

VKITTI	DDAD	Abs Rel ↓	Sq Rel ↓	RMSE ↓	$\delta_{1.25}$ ↑
✓		0.282	5.025	15.281	0.572
✓	✓	0.163	3.313	11.580	0.809
	✓	0.162	3.019	11.408	0.811

Table 4. **Refinement network training comparison.** We train $\mathcal{D}_\theta$ as described in the main paper with sparsified input depth and train $\mathcal{D}_\omega$ purely for monocular depth estimation without prior geometric depth input. We evaluate both networks on monocular depth estimation *without geometric depth input* on DDAD.

$\mathcal{D}_\omega(\mathbf{I}_t^c)$	$\mathcal{D}_\theta(\mathbf{I}_t^c, \mathbf{0}, \mathbf{0})$	Abs Rel ↓	Sq Rel ↓	RMSE ↓	$\delta_{1.25}$ ↑
✓		0.204	3.583	12.652	0.723
	✓	0.211	3.806	12.668	0.715

tions, but is at the same time very noisy, especially in low-textured areas and for dynamic objects. Our full method demonstrates the best performance.

In Fig. 4, we show a comparison of our method to the state-of-the-art approach SurroundDepth [8] on the NuScenes dataset. We observe that our approach produces significantly sharper and more accurate depth maps for all three of the examples.

Finally, we show additional comparison on accumulated point clouds on DDAD in Fig. 5 and on NuScenes in Fig. 6. Our method produces significantly more consistent 3D reconstructions than competing methods, as can be seen by the more consistent reconstruction of road markings, sidewalks, and trucks in Fig. 5. In Fig. 6 we observe that our method approaches the 3D reconstruction accuracy of the LiDAR-based 3D reconstruction.

Runtime breakdown and memory consumption. In Tab. 5 we show a component-wise breakdown of time-complexity and measured runtime of our approach. Compared to [7], we tackle a more complex scenario with six images per timestep. This leads to many more possible edges in the co-visibility graph, increasing the computational burden. Thus, the runtimes are slower than reported in [7]. However, Tab. 5 shows that runtime is dominated by the GRU, which scales with the number of edges $|\mathcal{E}|$. The breakdown and our observed $10\times$ runtime improvement are both at test time. The peak GPU memory consumption in inference with our parameter setting is 6.08 GB ($\sim$61 MB per edge).

Table 5. **Inference runtime analysis.** We list the runtime of each component of our system during inference.

Component	Complexity	Runtime [ms]	Percent [%]
Feature encoder	$\mathcal{O}(\lvert C\rvert)$	5	1.4
Context encoder	$\mathcal{O}(\lvert C\rvert)$	5	1.4
Create corr. volumes	$\mathcal{O}(\lvert\mathcal{E}_{new}\rvert)$	13	3.7
Corr. volume sampling	$\mathcal{O}(n_{\text{iter}} \cdot \lvert\mathcal{E}\rvert)$	31	8.8
GRU steps	$\mathcal{O}(n_{\text{iter}} \cdot \lvert\mathcal{E}\rvert)$	196	55.4
Multi-cam. DBA steps	$\mathcal{O}(n_{\text{iter}})$	1	0.3
Completion	$\mathcal{O}(\lvert\mathcal{V}\rvert)$	98	27.7
Others	-	5	1.4
Total		354	100

Limitations. The components of our system rely on deep neural networks with downsampling operations. This means a large chunk of the computation will happen at a lower resolution. While this provides a computational advantage, it also comes at the cost of losing high-frequency details that are important for thin structures like fences. In Fig. 2, we show an example of this phenomenon where our depth estimate results in a large error because there is an ambiguity between the background and the foreground fence. Similarly, other thin structures like poles could be missed, especially if they are far away.

3. Multi-Camera DBA

We provide a detailed derivation of our multi-camera DBA. The goal of the bundle adjustment is to minimize the energy function E, *i.e.* align edge-wise relative poses $\mathbf{G}_{ij}$ and frame-wise depth $\mathbf{d}_i$ whose reprojection minimizes the Mahalanobis distance to the estimated flow $\mathbf{f}_{ij}$ over all edges $\mathcal{E}$ in the co-visibility graph. In the following, let H

Figure 2. **Illustration of thin structures.** We depict an example frame, prediction, and error map from the NuScenes dataset. The fence in the picture poses a problem for our depth estimator since it is partially transparent. Further, since behind the fence, there is a large free space, we observe a large Sq. Rel. score for the areas that are predicted as background.

and W be the depth map's height and width and $T = |\mathcal{V}|/C$ the number of timesteps we consider, where C is the number of cameras.

$$E = \sum_{(i,j)\in\mathcal{E}} \left\| \mathbf{p}_{ij} - \Pi_{c_j}(\mathbf{G}_{ij} \circ \Pi_{c_i}^{-1}(\mathbf{d}_i)) \right\|_{\mathbf{\Sigma}_{ij}}^2 \tag{5}$$

With $\mathbf{\Sigma}_{ij} = \mathrm{diag}(\mathbf{w}_{ij})$, $\mathbf{p}_{ij} = \mathbf{x}_i + \mathbf{f}_{ij}$ and

$$\mathbf{G}_{ij} = \left(\mathbf{P}_{t_j}\mathbf{T}_{c_j}\right)^{-1}\mathbf{P}_{t_i}\mathbf{T}_{c_i} \tag{6}$$

where $\mathbf{T}_{c_i}$ and $\mathbf{T}_{c_j}$ are known constants and $\mathbf{P}_{t_i}$ and $\mathbf{P}_{t_j}$ are optimization variables. This will lead to updates

$$\begin{aligned} \mathbf{P}^{(k)} &= \exp(\delta\xi)\mathbf{P}^{(k-1)} \\ \mathbf{d}^{(k)} &= \mathbf{d} + \mathbf{d}^{(k-1)} \end{aligned} \tag{7}$$

where $\delta\xi$ and $\delta\mathbf{d}$ are the solution to the normal equation

$$\mathbf{J}^\top\mathbf{\Sigma}\mathbf{J}\cdot\begin{bmatrix}\delta\xi\\ \delta\mathbf{d}\end{bmatrix} = -\mathbf{J}^\top\mathbf{\Sigma}\mathbf{r} \tag{8}$$

where $\mathbf{J} \in \mathbb{R}^{|\mathcal{E}|\cdot H\cdot W\cdot 2\times(T\cdot 6+|\mathcal{V}|\cdot H\cdot W)}$ is the Jacobian of the residuals w.r.t. all optimization variables and $\mathbf{r} \in \mathbb{R}^{|\mathcal{E}|\cdot H\cdot W\cdot 2}$ is the vector of all residuals

3.1. Pose - Depth Decomposition

We can make three observations. First, pose $\mathbf{P}_k$ only appears in $\mathbf{r}_{ij}$ if k is either i or j and $t_i \neq t_j$. Second, there are no loops, thus $i \neq j$. Third, depth $\mathbf{d}_k$ only appears in $\mathbf{r}_{ij}$ if $k = i$, therefore $\frac{\partial \mathbf{r}_{ij}}{\partial \mathbf{d}_k} = 0\ \forall k \neq i$.

Let us now consider a single edge $(i,j) \in \mathcal{E}$ with $\mathbf{r}_{ij} \in \mathbb{R}^{H\cdot W\cdot 2}$ the residuals and $\mathbf{J}_{ij} \in \mathbb{R}^{H\cdot W\cdot 2\times(12+H\cdot W)}$ the Jacobian w.r.t. all optimization variables. We can decompose the Jacobian into its components, *i.e.* $\mathbf{J}_{ij} = \begin{bmatrix}\mathbf{J}_{\xi_i} + \mathbf{J}_{\xi_j} & \mathbf{J}_{\mathbf{d}_i}\end{bmatrix}$ where $\mathbf{J}_{\mathbf{d}_i}$ is diagonal. Now, when only considering the aforementioned edge, Eq. 8 can be written as

$$\begin{bmatrix}\mathbf{B}_{ii} & \mathbf{B}_{ij} & \mathbf{E}_{ii}\\ \mathbf{B}_{ji} & \mathbf{B}_{jj} & \mathbf{E}_{ji}\\ \mathbf{E}_{ii}^\top & \mathbf{E}_{ji}^\top & \mathbf{C}_i\end{bmatrix}\begin{bmatrix}\delta\xi_i\\ \delta\xi_j\\ \delta\mathbf{d}_i\end{bmatrix} = \begin{bmatrix}\mathbf{v}_i\\ \mathbf{v}_j\\ \mathbf{w}_i\end{bmatrix} \tag{9}$$

with

$$\begin{aligned} \mathbf{B}_{ii} &= \mathbf{J}_{\xi_i}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{J}_{\xi_i} & \mathbf{E}_{ii} &= \mathbf{J}_{\xi_i}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{J}_{\mathbf{d}_i}\\ \mathbf{B}_{ij} &= \mathbf{J}_{\xi_i}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{J}_{\xi_j} & \mathbf{E}_{ji} &= \mathbf{J}_{\xi_j}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{J}_{\mathbf{d}_i}\\ \mathbf{B}_{ji} &= \mathbf{J}_{\xi_j}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{J}_{\xi_i} & \mathbf{v}_i &= -\mathbf{J}_{\xi_i}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{r}_{ij}\\ \mathbf{B}_{jj} &= \mathbf{J}_{\xi_j}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{J}_{\xi_j} & \mathbf{v}_j &= -\mathbf{J}_{\xi_j}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{r}_{ij}\\ \mathbf{C}_i &= \mathbf{J}_{\mathbf{d}_i}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{J}_{\mathbf{d}_i} & \mathbf{w}_i &= -\mathbf{J}_{\mathbf{d}_i}^\top\mathbf{\Sigma}_{\mathbf{r}_{ij}}\mathbf{r}_{ij} \end{aligned} \tag{10}$$

We now consider again all edges $\mathcal{E}$. Because the energy function in Eq. 5 is the sum of the energies for all edges, we can apply the sum rule for derivatives. Thus, we can combine the components of the normal equation. Since the block matrices from Eq. 10 are dependent on their respective residual $\mathbf{r}_{ij}$, we can combine the blocks via a scattered sum

$$\begin{aligned} \mathbf{B} &= \sum_{(i,j)\in\mathcal{E}} \Phi_{t_it_i}^{T\times T}\left[\mathbf{B}_{ii}(\mathbf{r}_{ij})\right] + \Phi_{t_it_j}^{T\times T}\left[\mathbf{B}_{ij}(\mathbf{r}_{ij})\right] \\ &\quad + \Phi_{t_jt_i}^{T\times T}\left[\mathbf{B}_{ji}(\mathbf{r}_{ij})\right] + \Phi_{t_jt_j}^{T\times T}\left[\mathbf{B}_{jj}(\mathbf{r}_{ij})\right] \\ \mathbf{E} &= \sum_{(i,j)\in\mathcal{E}} \Phi_{t_ii}^{T\times|\mathcal{V}|}\left[\mathbf{E}_{ii}(\mathbf{r}_{ij})\right] + \Phi_{t_ji}^{T\times|\mathcal{V}|}\left[\mathbf{E}_{ji}(\mathbf{r}_{ij})\right] \\ \mathbf{C} &= \sum_{(i,j)\in\mathcal{E}} \Phi_{ii}^{|\mathcal{V}|\times|\mathcal{V}|}\left[\mathbf{C}_i(\mathbf{r}_{ij})\right] \\ \mathbf{v} &= \sum_{(i,j)\in\mathcal{E}} \Phi_{t_i}^{T\times 1}\left[\mathbf{v}_i(\mathbf{r}_{ij})\right] + \Phi_{t_j}^{T\times 1}\left[\mathbf{v}_j(\mathbf{r}_{ij})\right] \\ \mathbf{w} &= \sum_{(i,j)\in\mathcal{E}} \Phi_{i}^{|\mathcal{V}|\times 1}\left[\mathbf{w}_i(\mathbf{r}_{ij})\right] \end{aligned} \tag{11}$$

where $\Phi_{mn}^{M\times N} : \mathbb{R}^{U\times V} \rightarrow \mathbb{R}^{M\cdot U\times N\cdot V}$ is the function which maps a block matrix to row $m \in \{1,...,M\}$ and column $n \in \{1,...,N\}$ while the other elements are set to zero. To improve convergence, the Levenberg-Marquardt method is used on $\mathbf{C}$, leading to

$$\begin{bmatrix}\mathbf{B} & \mathbf{E}\\ \mathbf{E}^\top & \mathbf{C}+\lambda\mathbf{I}\end{bmatrix}\begin{bmatrix}\delta\xi\\ \delta\mathbf{d}\end{bmatrix} = \begin{bmatrix}\mathbf{v}\\ \mathbf{w}\end{bmatrix} \tag{12}$$

With $\mathbf{I}$ the identity matrix and pixel-wise damping factor λ is predicted by the GRU (see Sec. 1) for each node i. We observe that $\mathbf{C}$ is diagonal. We can use the Schur complement to solve Eq. 12 efficiently, due to $(\mathbf{C}+\lambda\mathbf{I})^{-1}$ being very easy to invert. Thus the updates are given by

$$\begin{aligned} \mathbf{S} &= \mathbf{B} - \mathbf{E}\mathbf{C}^{-1}\mathbf{E}^\top \\ \delta\xi &= \mathbf{S}^{-1}(\mathbf{v} - \mathbf{E}\mathbf{C}^{-1}\mathbf{w}) \\ \delta\mathbf{d} &= (\mathbf{C}+\lambda\mathbf{I})^{-1}(\mathbf{w} - \mathbf{E}^\top\delta\xi) \end{aligned} \tag{13}$$

Lastly, it can be shown that $\mathbf{S} \succ 0$, thus the Cholesky-decomposition can be used to efficiently solve for $\mathbf{S}^{-1}$.

3.2. Jacobians

Given the decomposition above, we now define the Jacobians $\mathbf{J}_{\xi_i}$, $\mathbf{J}_{\xi_j}$, and $\mathbf{J}_{\mathbf{d}_i}$. Let us consider a single depth d of node i at location (u, v). The residual is given by

$$\mathbf{r}_{ij,uv} = \mathbf{p}_{ij,uv} - \hat{\mathbf{p}}_{ij,uv} \in \mathbb{R}^2 \tag{14}$$

where $\hat{\mathbf{p}}_{ij,uv} = \Pi_{c_j}(\mathbf{G}_{ij} \circ \Pi_{c_i}^{-1}(d))$. We further define the 3D point corresponding to pixel (u, v) as $\mathbf{X} = \begin{bmatrix} X & Y & Z & W \end{bmatrix}^\top$ which is given by $\mathbf{X} = \Pi_{c_i}^{-1}(d)$ and the transformed point $\mathbf{X}' = \mathbf{G}_{ij}\mathbf{X}$. We can thus define the projection and unprojection operators for pinhole cameras c_i and c_j

$$\Pi_{c_j}(\mathbf{X}') = \begin{bmatrix} f_{c_j}^x \frac{X'}{Z'} + c_{c_j}^x \\ f_{c_j}^y \frac{Y'}{Z'} + c_{c_j}^y \end{bmatrix} \tag{15}$$

$$\Pi_{c_i}^{-1}(d) = \begin{bmatrix} \frac{u - c_{c_i}^x}{f_{c_i}^x} \\ \frac{v - c_{c_i}^y}{f_{c_i}^y} \\ 1 \\ d \end{bmatrix} \tag{16}$$

where f_c^x, f_c^y are the camera c's focal lengths in x and y direction, c_c^x, c_c^y are the respective principal points.

Depth The Jacobian $\mathbf{J}_d$ w.r.t. depth d is defined as

$$\begin{aligned} \mathbf{J}_d &= \frac{\partial \mathbf{r}_{ij}}{\partial d} = -\frac{\partial \hat{\mathbf{p}}_{ij}}{\partial d} = -\frac{\partial \Pi_{c_j}(\mathbf{X}')}{\partial \mathbf{X}'} \frac{\partial \mathbf{X}'}{\partial d} \\ &= -\frac{\partial \Pi_{c_j}(\mathbf{X}')}{\partial \mathbf{X}'} \mathbf{G}_{ij} \frac{\partial \Pi_{c_i}^{-1}(d)}{\partial d} \end{aligned} \tag{17}$$

$$\frac{\partial \Pi_{c_j}(\mathbf{X}')}{\partial \mathbf{X}'} = \begin{bmatrix} f_{c_j}^x \frac{1}{Z'} & 0 & -f_{c_j}^x \frac{X'}{Z'^2} & 0 \\ 0 & f_{c_j}^y \frac{1}{Z'} & -f_{c_j}^y \frac{Y'}{Z'^2} & 0 \end{bmatrix} \tag{18}$$

$$\frac{\partial \Pi_{c_i}^{-1}(d)}{\partial d} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{19}$$

Pose The Jacobian $\mathbf{J}_\xi$ w.r.t. ξ where $\xi \in \mathfrak{se}(3)$ is either ξ_i or ξ_j.

$$\mathbf{J}_\xi = \frac{\partial \mathbf{r}_{ij}}{\partial \xi} = -\frac{\partial \hat{\mathbf{p}}_{ij}}{\partial \xi} = -\frac{\partial \Pi_{c_j}(\mathbf{X}')}{\partial \mathbf{X}'} \frac{\partial \mathbf{X}'}{\partial \xi} \tag{20}$$

The partial derivative $\frac{\partial \Pi_{c_j}(\mathbf{X}')}{\partial \mathbf{X}'}$ has been derived in Eq. 18. For $\frac{\partial \mathbf{G}_{ij}}{\partial \xi}$, we can again decompose $\mathbf{G}_{ij}$ into the static parts $\mathbf{T}_{c_i}$ and $\mathbf{T}_{c_j}$ and the unknown, to be optimized parts, $\mathbf{P}_{t_i}$ and $\mathbf{P}_{t_j}$

$$\mathbf{X}' = (\mathbf{P}_{t_j}\mathbf{T}_{c_j})^{-1}\mathbf{P}_{c_i}\mathbf{T}_{c_i}\mathbf{X} \tag{21}$$

First, similar to Eq. 39 to 44 in [2], for $\mathbf{A} = \mathbf{A}_1(\mathbf{A}_0)^{-1}$ we can write

$$\begin{aligned} \frac{\partial \mathbf{A}}{\partial \mathbf{A}_0} &= \frac{\partial \log\left(\mathbf{A}_1(\exp(\xi)\mathbf{A}_0)^{-1}(\mathbf{A}_1\mathbf{A}_0^{-1})^{-1}\right)}{\partial \xi}|_{\xi=0} \\ &= \frac{\partial}{\partial \xi}|_{\xi=0}\left[\log\left(\mathbf{A}_1\mathbf{A}_0^{-1}\exp(-\xi)\mathbf{A}_0\mathbf{A}_1^{-1}\right)\right] \\ &= \frac{\partial}{\partial \xi}|_{\xi=0}\left[\log\left(\exp(-\operatorname{Adj}_{\mathbf{A}_1\mathbf{A}_0^{-1}}\xi)\mathbf{A}_1\mathbf{A}_0^{-1}\mathbf{A}_0\mathbf{A}_1^{-1}\right)\right] \\ &= \frac{\partial}{\partial \xi}|_{\xi=0}\left[\log\left(\exp(-\operatorname{Adj}_{\mathbf{A}_1\mathbf{A}_0^{-1}}\xi)\right)\right] \\ &= \frac{\partial}{\partial \xi}|_{\xi=0}\left[-\operatorname{Adj}_{\mathbf{A}_1\mathbf{A}_0^{-1}}\xi\right] \\ &= -\operatorname{Adj}_{\mathbf{A}_1\mathbf{A}_0^{-1}} \end{aligned} \tag{22}$$

In the following, we omit the explicit perturbation around $\xi = 0$. Let now $\mathfrak{G}_1, ..., \mathfrak{G}_6 \in \mathbb{R}^{4\times4}$ be generators as defined in Eq. 65 in [2]. With the chain rule and Eq. 94 in [2] and Eq. 22 above, the derivative w.r.t. the pose of incoming node j is given by

$$\begin{aligned} \frac{\partial \mathbf{X}'}{\partial \xi_j} &= \frac{\partial}{\partial \xi_j}\left[(\exp(\xi_j)\mathbf{P}_{t_j}\mathbf{T}_{c_j})^{-1}\mathbf{P}_{t_i}\mathbf{T}_{c_i}\mathbf{X}\right] \\ &= \frac{\partial}{\partial \xi_j}[\underbrace{\mathbf{T}_{c_j}^{-1}(\exp(\xi_j)\mathbf{P}_{t_j})^{-1}}_{\mathbf{A}}\underbrace{\mathbf{P}_{t_i}\mathbf{T}_{c_i}\mathbf{X}}_{\mathbf{X}^w}] \\ &= \frac{\partial}{\partial \mathbf{A}}\left[\mathbf{A}\mathbf{X}^w\right] \cdot \frac{\partial}{\partial \xi_j}[\underbrace{\mathbf{T}_{c_j}^{-1}}_{\mathbf{A}_1}\underbrace{(\exp(\xi_j)\mathbf{P}_{t_j})^{-1}}_{\mathbf{A}_0}] \\ &= \frac{\partial}{\partial \mathbf{A}}[\mathbf{A}\mathbf{X}^w] \cdot \frac{\partial}{\partial \mathbf{A}_0}[\mathbf{A}_1\mathbf{A}_0^{-1}] \\ &= \begin{bmatrix}\mathfrak{G}_1\mathbf{X}^w & ... & \mathfrak{G}_6\mathbf{X}^w\end{bmatrix} \cdot (-\operatorname{Adj}_{\mathbf{A}_1\mathbf{A}_0^{-1}}) \\ &= -\begin{bmatrix}\mathfrak{G}_1\mathbf{X}^w & ... & \mathfrak{G}_6\mathbf{X}^w\end{bmatrix} \cdot \operatorname{Adj}_{\mathbf{T}_{c_j}^{-1}\mathbf{P}_{t_j}^{-1}} \end{aligned} \tag{23}$$

Finally, with Eq. 94 and 97 in [2] and the derivative w.r.t. the pose of the outgoing node i

$$\begin{aligned} \frac{\partial \mathbf{X}'}{\partial \xi_i} &= \frac{\partial}{\partial \xi_i}[\underbrace{(\mathbf{P}_{t_j}\mathbf{T}_{c_j})^{-1}\exp(\xi_i)\mathbf{P}_{t_i}}_{\mathbf{A}}\underbrace{\mathbf{T}_{c_i}\mathbf{X}}_{\mathbf{X}^{t_i}}] \\ &= \frac{\partial}{\partial \mathbf{A}}[\mathbf{A}\mathbf{X}^{t_i}] \cdot \frac{\partial}{\partial \xi_i}[(\mathbf{P}_{t_j}\mathbf{T}_{c_j})^{-1}\exp(\xi_i)\mathbf{P}_{t_i}] \\ &= \begin{bmatrix}\mathfrak{G}_1\mathbf{X}^{t_i} & ... & \mathfrak{G}_6\mathbf{X}^{t_i}\end{bmatrix} \cdot \operatorname{Adj}_{\mathbf{T}_{c_j}^{-1}\mathbf{P}_{t_j}^{-1}} \end{aligned} \tag{24}$$

Finally, we compose the component-wise Jacobians above into the Jacobian $\mathbf{J}$ as stated in Sec. 3.1. We can now use $\mathbf{J}$ to compute the updates $\delta\xi$ and δd in Eq. 8.

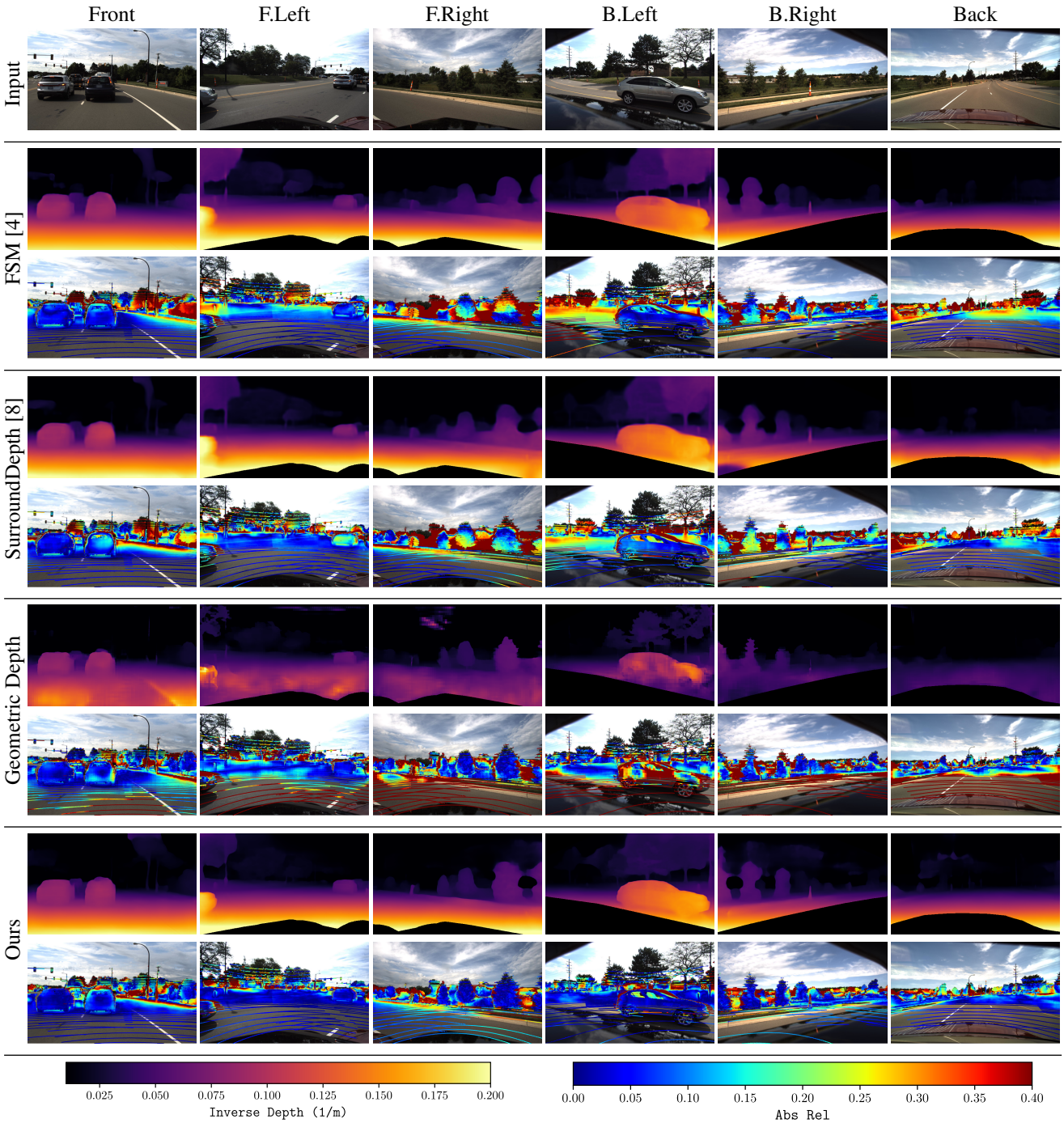

Figure 3. **Qualitative comparison on DDAD.** We compare existing works to both our geometric and refined depth estimates. Especially for the side views, existing works struggle to obtain accurate depth. The geometric depth produces many accurate depth predictions, but contains many noisy points, especially in low-textured areas and for dynamic objects. Our full method demonstrates the best performance.

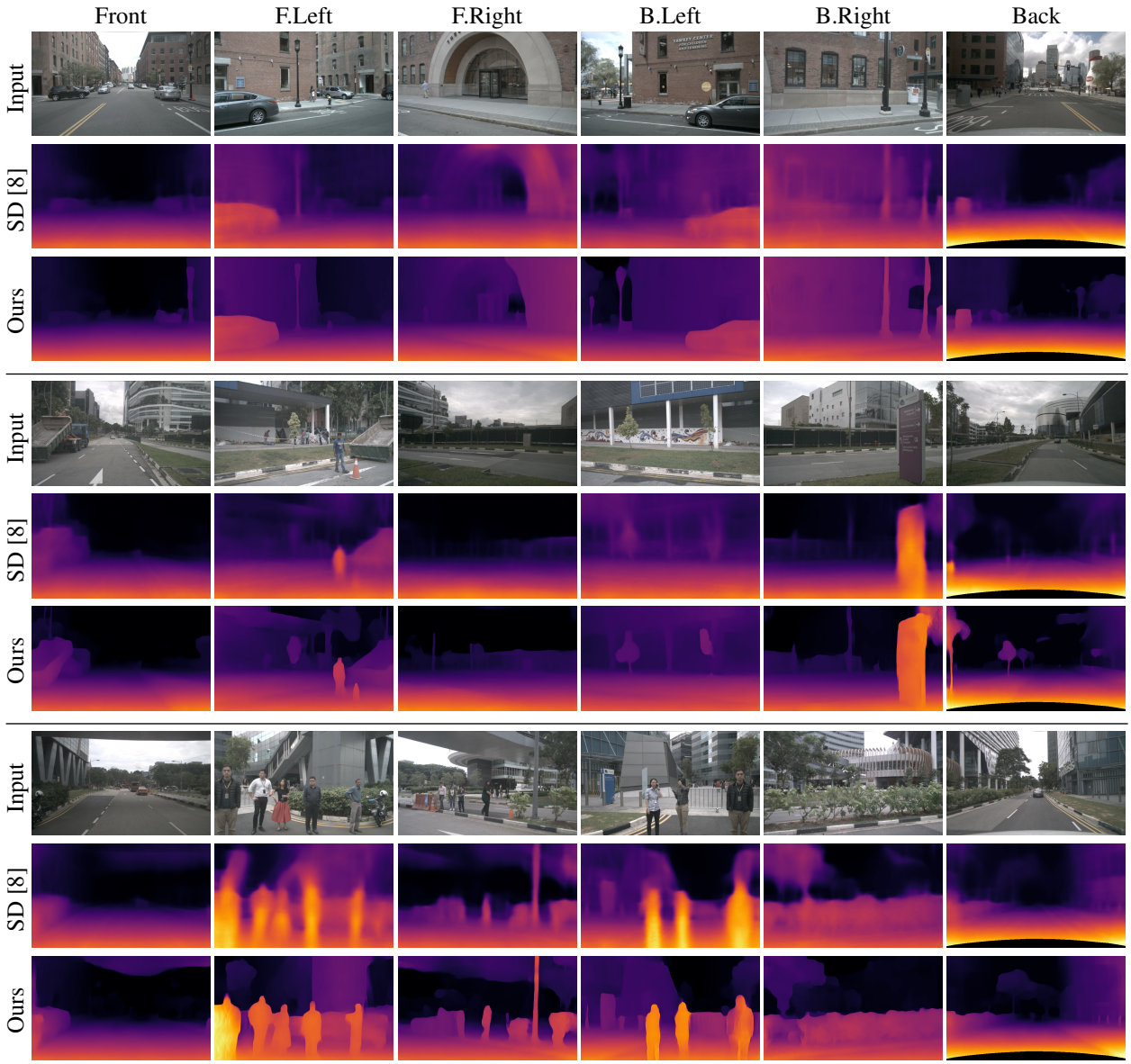

Figure 4. **Qualitative comparison on NuScenes**. We show a comparison of depth maps from our method to the depth maps of the state-of-the-art approach SurroundDepth [8]. We observe that our approach produces significantly sharper and more accurate depth predictions.

Figure 5. **Qualitative comparison of 3D reconstructions on DDAD.** We show the 3D reconstructions of our method compared to SurroundDepth [8] and FSM [4]. Additionally, we plot the ground-truh LiDAR 3D reconstruction at the top. The ego-vehicle trajectory is marked in red. We observe that our method produces significantly more consistent and accurate 3D reconstructions than competing methods, as can be seen when focusing on the trucks (right) and the street markings and pedestrians (left).

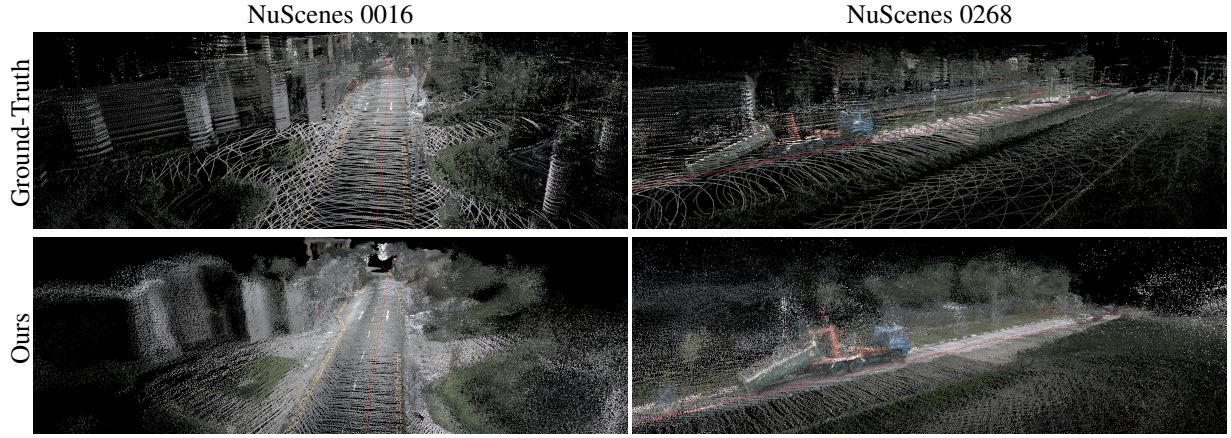

Figure 6. **Qualitative 3D reconstruction results on NuScenes.** We show our 3D reconstruction results alongside the LiDAR ground-truth 3D reconstruction. The ego-vehicle trajectory is marked in red. We observe that our method yields similarly consistent 3D reconstruction results as the LiDAR ground-truth while being denser.

References

[1] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual KITTI 2. *CoRR*, 2020.

[2] Ethan Eade. Lie groups for 2d and 3d transformations. *URL http://ethaneade. com/lie. pdf, revised Dec*, 2013.

[3] Vitor Guizilini, Rui Hou, Jie Li, Rares Ambrus, and Adrien Gaidon. Semantically-guided representation learning for self-supervised monocular depth. In *ICLR*, 2020.

[4] Vitor Guizilini, Igor Vasiljevic, Rares Ambrus, Greg Shakhnarovich, and Adrien Gaidon. Full surround monodepth from multiple cameras. In *RA-L*, 2022.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[6] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*, 2020.

[7] Zachary Teed and Jia Deng. DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras. *NeurIPS*, 2021.

[8] Yi Wei, Linqing Zhao, Wenzhao Zheng, Zheng Zhu, Yongming Rao, Guan Huang, Jiwen Lu, and Jie Zhou. Surrounddepth: Entangling surrounding views for self-supervised multi-camera depth estimation. In *CoRL*, 2022.