Non-parametric Depth Distribution Modelling based Depth Inference for Multi-view Stereo Supplementary Material

Jiayu Yang[1,2], Jose M. Alvarez[2], Miaomiao Liu[1]
[1]Australian National University, [2]NVIDIA
{jiayu.yang, miaomiao.liu}@anu.edu.au, josea@nvidia.com

In this supplementary material, we first introduce the sparse cost aggregation network in Section 1. In Section 2, we show additional qualitative results by our method for different scans on the DTU dataset. In Section 3, we show additional qualitative results by our method for different scenes on the Tanks and Temples dataset. In Section 4, we provide details of the segmentation of $\{R0, R1, \cdots, R4\}$ based on Laplacian pyramid. In Section 5, we provide comparison of runtime and memory with existing methods. In Section 6, we provide additional ablation studies including training the model by applying the KL-divergence as loss function, validation of the top-k hypotheses selection and applying unsupervised training on the proposed model.

1. Sparse Cost Aggregation Network

In this section, we provide details of the sparse cost aggregation network. As introduced in the main paper, the sparse cost volume C can not be efficiently aggregated by regular dense 3D convolutions. We build a sparse cost aggregation network to aggregate cost utilizing the rigid spatial relation stored in p_k. Specifically, the basic block of our network consists of three layers of sparse 3D convolution with factorized kernels on each dimension [29], a sparse batch normalization layer and a sparse ReLU activation. The factorized kernels follow the kernel factorization in [28] to improve efficiency and reduce memory consumption. Details of the basic block are shown in Tab. 8a. Using the basic block, we build a sparse 3D U-Net of 4 levels for cost aggregation. The output of the network is normalized by Softmax along the dimension of depth hypothesis to represent probability. Details are in Tab. 8b. The sparse cost aggregation network is implemented using the *Torchsparse* [29] library.

2. More qualitative results on DTU dataset

Fig. 13 shows additional qualitative results on DTU dataset. Our method based on non-parametric depth distribution modeling can achieve more accurate depth estimation on small objects (see the first row of Fig. 13) and boundary regions.

3. More qualitative results on Tanks and Temples dataset

Fig. 12 shows additional qualitative results on Tanks and Temples dataset [27]. We compare to the point cloud generated by the unimodal based method CVP-MVSNet [32]. Our method based on non-parametric depth distribution modeling can achieve more accurate depth estimation on small objects and boundary regions.

4. Segmentation using Laplacian pyramid

In this section we provide more details of using Laplacian pyramid for segmenting the ground-truth depth. Assume $\text{UP}_q(D)$ and $\text{DOWN}_q(D)$ denote upsampling and downsampling an input depth map D for q times. For example, $q = 0$ defines the operation which directly returns its input unchanged. We first build a depth map pyramid $\{\mathbf{D}^q_{gt}\}^Q_{q=0}$ of $Q+1$ levels by repeatedly downsampling the ground-truth depth map $\mathbf{D}^0_{gt}$.

$$\mathbf{D}^q_{gt} = \text{DOWN}_q(\mathbf{D}^0_{gt}) \tag{10}$$

We then build the Laplacian pyramid of depth denoted as $\{\text{Lap}^q_{gt}\}^Q_{q=0}$ by taking the absolute difference between adjacent levels in $\{\mathbf{D}^q_{gt}\}^Q_{q=0}$, where the lower level depth is upsampled to match the size of the upper one.

$$\text{Lap}^q_{gt} = |\mathbf{D}^q_{gt} - \text{UP}_1(\mathbf{D}^{q+1}_{gt})| \tag{11}$$

Each level of this Laplacian pyramid contains depth structures present at a particular scale, which we use as a mask to segment the depth map into different smoothness regions. Specifically, we generate the mask by finding regions with significant depth residual on certain level. For each pixel p, its mask R^q_p is computed by a threshold τ.

$$R_p^q = \begin{cases} 1 & \text{if } \text{Lap}_{gt,p}^q > \tau \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

The τ is set as a very small value. We set τ to be $0.5mm$ according to 0.1% of the depth range $[425, 935]mm$ on DTU dataset. Fig. 11 shows more visualization of the segmentation result.

5. Computational effectiveness

Our model adopts sparse convolution layers from [16] for sparse cost volume aggregation, which is not fully optimized in terms of efficiency. But still our specifically designed sparse cost aggregation network can achieve overall computational effectiveness similar to existing 3D convolution based methods and achieved best *overall* score on DTU dataset(see Fig. 10). Our method can achieve better performance and computational effectiveness along with the improvement of sparse convolution in the future.

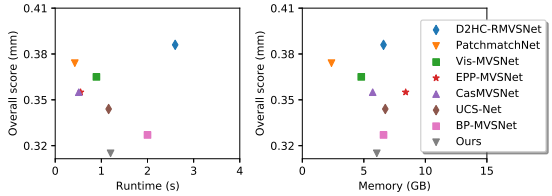

Figure 10. **DTU dataset.** Comparison of runtime and memory consumption.

6. Ablation Study

6.1. Effect of sparse cost aggregation

In Table 5, we evaluate the depth estimation accuracy on different pixel sets based on the textureness or the depth gradient. For the former, we set a threshold on the image variance to 0.0005. The latter, with a threshold of $1mm$. Statistically, there is a boost of about 1.93% in the point cloud size compared to the method without sparse cost aggregation. This improvement is the result of more 3D points been preserved by the point cloud fusion process via the geometric consistency check. More points preserved leads to improvements in the *completeness* metric, reflecting that the distribution of the estimated point cloud better follows the ground truth.

Method	Texture		Depth smoothness	
	Low	High	Non-boundary	Boundary
Baseline (Uni-modal)	15.801	2.689	5.915	8.359
+ Sparse Cost Agg.	**15.352**	**2.324**	**5.829**	**7.961**

Table 5. Average depth error (mm) on different regions

6.2. Validation of top-k hypothesis selection

In this section we provide the validation experiments of the parameter k for top-k hypotheses. To this end, we validate k by selecting top-k hypotheses from the approximated ground-truth depth distribution $\mathbf{P}_{gt,p}^l$ on each level and report the percentage of pixels on full resolution, the ground truth depths of which are covered by the selected k hypotheses on all levels. Results are shown in Tab. 6. Our chosen parameters $\{k^l\}_{l=0}^{L-1} = \{4, 8, 16\}$ which correspond to $\{M^l\}_{l=0}^{L} = \{8, 16, 32, 96\}$ can cover almost all pixels on the DTU dataset. Increasing k does not provide better coverage. Reducing k will cause 1.4% pixels not covered by the top-k selected hypothesis.

k	Covered pixels
$\{8, 16, 32\}$	99.99636%
$\{4, 8, 16\}$	99.99622%
$\{2, 4, 8\}$	98.52702%

Table 6. **DTU dataset.** Percentage of pixels, the ground truth depths of which are covered by top-k selected hypotheses using different k.

6.3. Using KL divergence as loss

In this section we provide additional results of using Kullback–Leibler divergence as loss to train our network on DTU dataset. As mentioned in the main paper, for each pixel p, we use the depth probability distribution $\mathbf{P}_{gt,p}^l$ approximated by the high-resolution depth map observations as ground-truth to supervise the estimated depth distribution $\mathbf{P}_p^l$. As a measure of the difference between two probability distributions, the Kullback–Leibler divergence can also be used as loss function. Specifically, for each pixel p on each level l, we use the KL divergence between the estimated depth distribution $\mathbf{P}_p^l$ and the approximated ground-truth depth distribution $\mathbf{P}_{gt,p}^l$ as loss.

$$\mathcal{L}_p^l = \text{KL}(\mathbf{P}_{gt,p}^l, \mathbf{P}_p^l) \tag{13}$$

Results are shown in Tab. 7. Comparing with the model trained by Binary Cross Entropy loss, using KL-divergence as loss function can achieve slightly better mean accuracy but slightly worse mean completeness.

Models	Acc.	Comp.	Overall
Ours (BCE)	0.3563	**0.2750**	**0.3156**
Ours (KL)	**0.3479**	0.2930	0.3205

Table 7. **DTU dataset.** Quantitative results of our model using different loss functions.

6.4. Unsupervised training

Here we explore unsupervised training for the proposed network. We follow [31] to use pseudo depth to generate the depth distributions as supervision. We ran experiments

on the DTU dataset using this setting and achieved *accuracy* 0.385, *completeness* 0.323 and *overall* 0.354, which outperform existing unsupervised MVS methods.

References

[27] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 2017. 1

[28] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016. 1

[29] Haotian* Tang, Zhijian* Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *ECCV*, 2020. 1

[30] Fangjinhua Wang, Silvano Galliani, Christoph Vogel, Pablo Speciale, and Marc Pollefeys. Patchmatchnet: Learned multi-view patchmatch stereo, 2021. 5

[31] Jiayu Yang, Jose Alvarez, and Miaomiao Liu. Self-supervised learning of depth inference for multi-view stereo. In *CVPR*, 2021. 2

[32] Jiayu Yang, Wei Mao, Jose M Alvarez, and Miaomiao Liu. Cost volume pyramid based depth inference for multi-view stereo. In *CVPR*, 2020. 1, 4

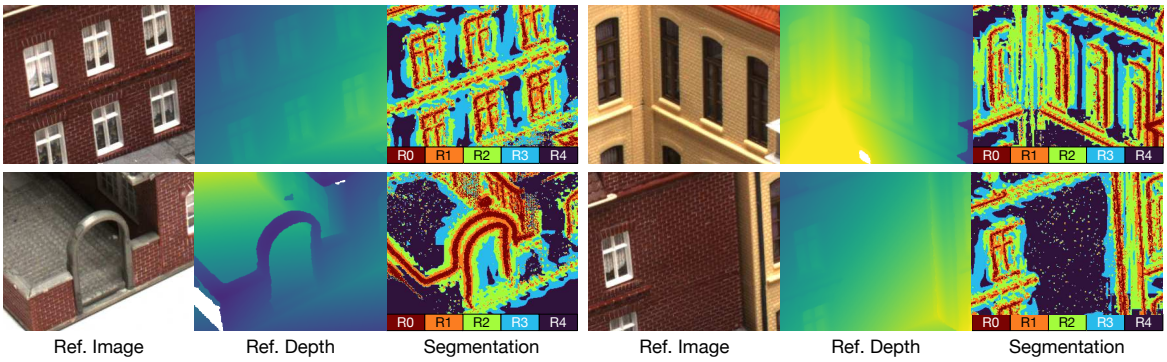

Figure 11. More visualization of depth map segmentation. We segment ground-truth depth map into five regions corresponding to different depth smoothness and evaluate depth estimation accuracy on each region. The R0 region contains pixels with most abrupt depth change, such as object boundaries or small objects.

Figure 12. **Tanks and Temples dataset.** More qualitative results on small objects and boundaries. Comparing with the unimodal based method CVP-MVSNet [32], our method based on non-parametric depth distribution modeling can achieve more accurate depth estimation on small objects and boundary regions.

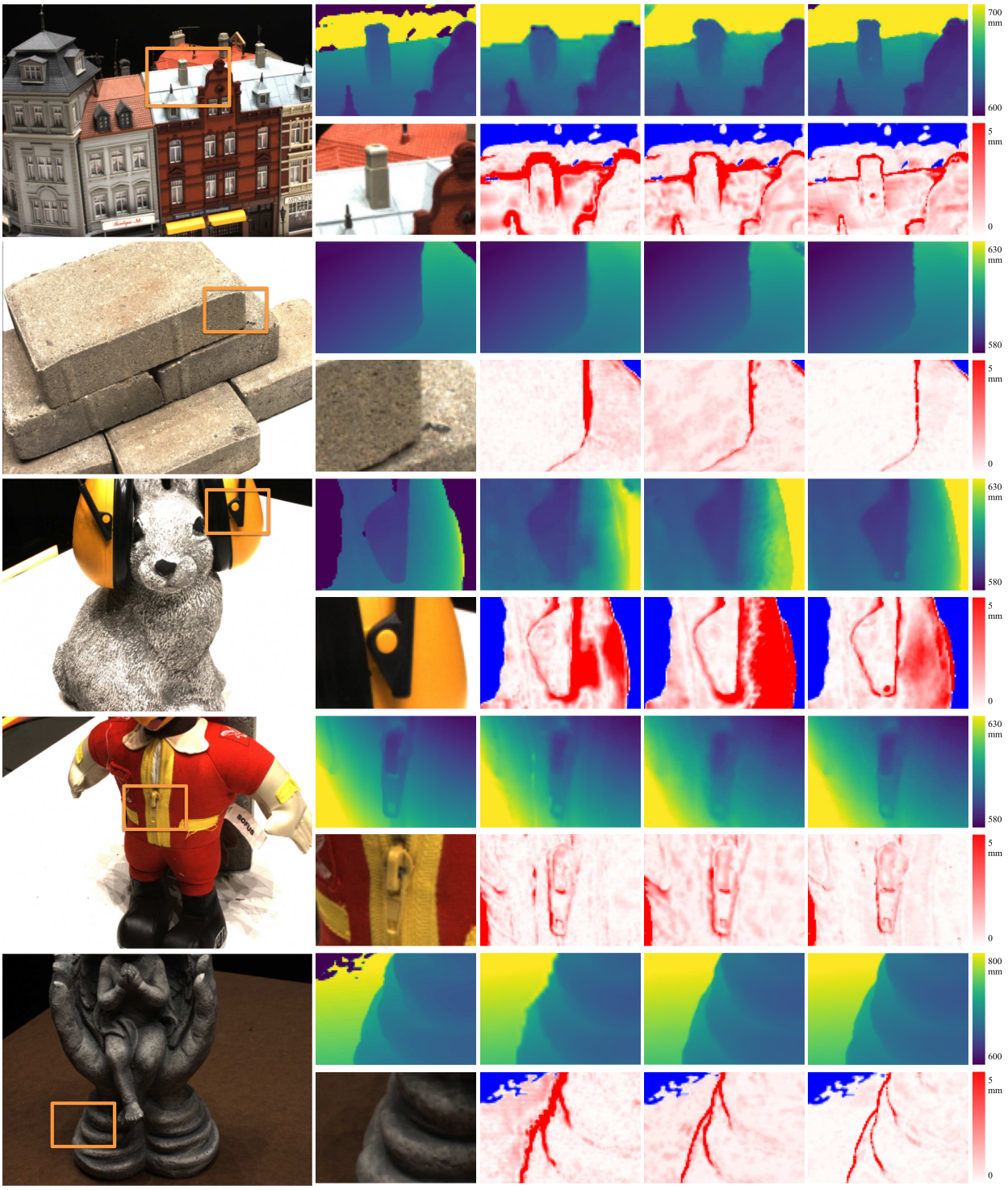

Figure 13. **DTU dataset.** More qualitative results on small objects and depth boundaries. Every upper row shows the ground-truth and estimated depth maps and under row shows the estimation error comparing with ground-truth depth map. Areas with no ground-truth are marked blue. Our method based on non-parametric depth distribution modeling is more accurate on small objects (row 1,3,4) and boundary regions (row 2,5).

Input	Input Size	Layer	Output	Output Size
(a) Sparse 3D Basic Block: ConvBnRelu3DSparseFactorize				
input	$K \times CH$	sparse conv3d, kernel_size=1x1x3, stride=1	-	$K \times CH$
-	$K \times CH$	sparse conv3d, kernel_size=1x3x1, stride=1	-	$K \times CH$
-	$K \times CH$	sparse conv3d, kernel_size=3x1x1, stride=1	-	$K \times CH$
-	$K \times CH$	sparse BatchNorm	-	$K \times CH$
-	$K \times CH$	sparse ReLU	output	$K \times CH$
(b) Sparse Cost Aggregation Network				
C^l	$K \times CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K \times CH$
-	$K \times CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K \times CH$
-	$K \times CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	c0	$K \times CH$
c0	$K \times CH$	sparse conv3d, kernel_size=2x2x2, stride=2	-	$K/4 \times 2CH$
-	$K/4 \times 2CH$	sparse BatchNorm	-	$K/4 \times 2CH$
-	$K/4 \times 2CH$	sparse ReLU	-	$K/4 \times 2CH$
-	$K/4 \times 2CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K/4 \times 2CH$
-	$K/4 \times 2CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	c1	$K/4 \times 2CH$
c1	$K/4 \times 2CH$	sparse conv3d, kernel_size=2x2x2, stride=2	-	$K/16 \times 4CH$
-	$K/16 \times 4CH$	sparse BatchNorm	-	$K/16 \times 4CH$
-	$K/16 \times 4CH$	sparse ReLU	-	$K/16 \times 4CH$
-	$K/16 \times 4CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K/16 \times 4CH$
-	$K/16 \times 4CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	c2	$K/16 \times 4CH$
c2	$K/16 \times 4CH$	sparse conv3d, kernel_size=2x2x2, stride=2	-	$K/64 \times 8CH$
-	$K/64 \times 8CH$	sparse BatchNorm	-	$K/64 \times 8CH$
-	$K/64 \times 8CH$	sparse ReLU	-	$K/64 \times 8CH$
-	$K/64 \times 8CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K/64 \times 8CH$
-	$K/64 \times 8CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	c3	$K/64 \times 8CH$
c3	$K/64 \times 8CH$	sparse conv3d, kernel_size=2x2x2, stride=2, transposed=True	-	$K/16 \times 4CH$
-	$K/16 \times 4CH$	sparse BatchNorm	-	$K/16 \times 4CH$
-	$K/16 \times 4CH$	sparse ReLU	-	$K/16 \times 4CH$
-	$K/16 \times 4CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K/16 \times 4CH$
-	$K/16 \times 4CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	c3up	$K/16 \times 4CH$
c2 + c3up	$K/16 \times 4CH$	sparse conv3d, kernel_size=2x2x2, stride=2, transposed=True	-	$K/4 \times 2CH$
-	$K/4 \times 2CH$	sparse BatchNorm	-	$K/4 \times 2CH$
-	$K/4 \times 2CH$	sparse ReLU	-	$K/4 \times 2CH$
-	$K/4 \times 2CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K/4 \times 2CH$
-	$K/4 \times 2CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	c2up	$K/4 \times 2CH$
c1 + c2up	$K/4 \times 2CH$	sparse conv3d, kernel_size=2x2x2, stride=2, transposed=True	-	$K \times CH$
-	$K \times CH$	sparse BatchNorm	-	$K \times CH$
-	$K \times CH$	sparse ReLU	-	$K \times CH$
-	$K \times CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K \times CH$
-	$K \times CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	c1up	$K \times CH$
c0 + c1up	$K \times CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K \times CH$
-	$K \times CH$	ConvBnRelu3DSparseFactorize, kernel_size=3, stride=1	-	$K \times CH$
-	$K \times CH$	sparse conv3d, kernel_size=1x1x3, stride=1	-	$K \times CH$
-	$K \times CH$	sparse conv3d, kernel_size=1x3x1, stride=1	-	$K \times CH$
-	$K \times CH$	sparse conv3d, kernel_size=3x1x1, stride=1	-	$K \times CH$
-	$K \times CH$	sparse conv3d, kernel_size=1x1x1, stride=1	-	$K \times 1$
-	$K \times 1$	Softmax	$\mathbf{P}^l$	$K \times 1$

Table 8. Details of the sparse cost aggregation network. Layers with unnamed input or output are sequential layers, where the input or output are directly connected to previous or following layer.