In this supplementary material, we first list the details of our implementation in section A. Second we conduct additional experiments including an analysis on our hyperparameter, runtime analysis, and an evaluation on a more challenging dataset, in section B. Third, we provide the details of the network architecture in section C. Finally, we provide qualitative visualization of our PointMBF in section D.

A. Implementation Details of PointMBF

Table 5 shows the implementation details of our PointMBF. The first nine lines are the same as those in LLT [67] and UR&R [14], while the last three lines colored in grey are our own settings.

Table 5. Implementation details of our PointMBF.

Batch size	8
Image size	128*128
Feature dimension	32
Number of correspondence k	200
Training epochs	12
Optimizer	Adam
Learning rate	1e-4
Momentum	0.9
Weight decay	1e-6
K_{v2g}, K_{g2v} for training	$K_{v2g} = 16, K_{g2v} = 1$
K_{v2g}, K_{g2v} for test	$K_{v2g} = 32, K_{g2v} = 1$
Use pre-trained weight for ResNet18	False

B. Additional Experiments

In this section, we conduct two experiments including analysis on hyperparameter K_{v2g} and runtime analysis. All the models in the experiments is trained on 3DMatch [73] and tested on ScanNet [11].

B.1. Effect of hyperparameter K_{v2g}

Hyperparameter K_{v2g} denotes the number of visual features embedded into each geometric feature. Here, we test its influence on registration. Limited by memory, we set it from 1 to 32.

The result is shown in Table 6. It can be seen that the performance of our method improves as K_{v2g} increases, but the trend of improvement gradually slows down. This is because that, as K_{v2g} increases, K_{v2g} gradually reaches the number of points or pixels in a corresponding region. We also find that even if we set K_{v2g} to 1, our method still outperforms the state-of-the-art method, LLT [67] in almost all metrics, which illustrates the effectiveness of our method.

B.2. Runtime Analysis

We conduct runtime analysis by comparing time overhead on each step of unsupervised RGB-D registration. Both our PointMBF and the competitor, UR&R are tested on an A40 graphic card with an Intel Xeon Platinum 8358P CPU. We report the mean and standard deviation of running time of each step.

The result is shown in Table 8. It can be seen that our multi-scale bidirectional fusion greatly improves performance by a large margin without adding much time overhead ($< 10ms$). Furthermore, the extra overhead on feature extraction is negligible compared to the overhead of correspondence estimation (main overhead).

B.3. Evaluation on ScanNet-SuperGlue

Our experiments in the main paper follow the settings of UR&R [14], in which view pairs are 20 frames apart. We find 20 frames apart is less challenging, making the effectiveness of our method less obvious. Specifically, over 99.8% of the ground truth rotation is under 45°, which makes ICP performs best under 45° threshold in Table 1. Moreover, too many easy cases also cause similar medians in Table 1, especially for chamfer errors. Therefore, we conduct a more challenging evaluation on ScanNet-SuperGlue.

ScanNet-SuperGlue is a dataset based on ScanNet [11], which is provided by SuperGlue [56]. It includes 1500 view pairs with average 480.8 frames apart. Our competitors include UR&R and a fusion-based method named CAT. CAT utilizes the same visual and geometric branches as our method and fuses visual features and geometric features using a concatenation operator at the final stage. All the above methods including our PointMBF are trained on 3DMatch [73] and tested on ScanNet-SuperGlue.

As shown in Table 7, our method outperforms the others by a large margin. Moreover, the median chamfer errors vary considerably because this experiment includes more hard cases.

C. Details of the Network Architecture

In this section, we provide details of the network architecture including feature extractor, point/pixel gathering for fusion, geometric fitting, keypoint, and differentiable rendering.

C.1. Feature extractor

Figure 6 shows the detailed architecture of the feature extractor in our PointMBF. As mentioned in section 3.1, we modify a ResNet18 [20] into our visual branch. The encoder consists of conv2_x, conv3_x, and conv4_x in ResNet18, and we remove the max pooling in original conv2_x. The decoder mainly consists of upsampling module, concatenation operator, and convblock i.e. shallow perception module. Our geometric branch has a symmetric architecture to the visual branch.

Table 6. Registration results under different hyperparameter K_{v2g}.

	Rotation (deg)					Translation (cm)					Chamfer (mm)				
	Accuracy↑			Error↓		Accuracy↑			Error↓		Accuracy↑			Error↓	
	5	10	45	Mean	Med.	5	10	25	Mean	Med.	1	5	10	Mean	Med.
LLT [67]	93.4	96.5	**98.8**	**2.5**	**0.8**	76.9	90.2	96.7	**5.5**	2.2	86.4	95.1	96.8	**4.6**	**0.1**
K_{v2g}=1	93.7	96.6	98.6	3.2	**0.8**	79.5	91.1	96.7	6.6	2.2	90.2	96.1	97.2	5.4	**0.1**
K_{v2g}=2	93.7	96.4	98.7	3.2	**0.8**	79.7	91.2	96.6	6.5	2.2	90.3	96.1	97.1	5.3	**0.1**
K_{v2g}=4	94.0	96.6	98.6	3.2	**0.8**	80.1	91.6	96.8	6.6	**2.1**	90.6	96.3	97.2	5.4	**0.1**
K_{v2g}=8	94.3	96.7	98.7	3.1	**0.8**	80.4	91.6	96.9	6.5	**2.1**	90.9	96.4	97.3	5.2	**0.1**
K_{v2g}=16	94.5	96.9	98.7	3.1	**0.8**	80.8	91.7	97.0	6.3	**2.1**	91.1	96.4	97.3	5.1	**0.1**
K_{v2g}=32	**94.6**	**97.0**	98.7	3.0	**0.8**	**81.0**	**92.0**	**97.1**	6.2	**2.1**	**91.3**	**96.6**	**97.4**	4.9	**0.1**

Table 7. Performance on ScanNet (Splitted by SuperGlue). CAT denotes fusion using direct concatenation.

	Rotation (deg)				Translation (cm)				Chamfer (mm)			
	Accuracy↑			Error↓	Accuracy↑			Error↓	Accuracy↑			Error↓
	5	10	45	Med.	5	10	25	Med.	1	5	10	Med.
UR&R [14]	36.0	49.0	82.3	10.5	18.5	29.7	48.3	26.9	24.6	41.3	48.8	11.1
CAT	47.1	56.8	81.1	6.2	27.7	41.2	55.5	17.7	40.5	54.1	59.7	3.2
Ours	**51.1**	**60.8**	**82.9**	**4.7**	**31.5**	**44.2**	**59.3**	**13.8**	**44.8**	**57.5**	**63.3**	**1.7**

Table 8. Runtime analysis.

	Time (ms)	
	UR&R [14]	Ours
Feature Extraction	21.94±20.92	31.92±8.97
Correspondence Estimation	247.88±23.22	255.94±22.95
Geometric Fitting	9.11±5.25	8.91±5.44
Rendering (Just for training)	8.32±6.73	8.10±6.28

C.2. Point/pixel gathering for fusion

During our feature extraction, we embed regional visual features into geometric features and regional geometric features into visual features. In this procedure, it's important to gather corresponding points/pixels for fusion. Here, we provide the details of the point/pixel gathering process, which is shown in Figure 4.

For visual-to-geometric fusion in the l-th layer, given a query point, we first determine its neighbor ball with radius R^l_{v2g}. Then we project this neighbor ball to the camera and the pixels falling in the projection region are selected as candidate pixels for fusion. After that, we filter the improper pixels in candidate pixels. There are two categories of inproper pixels. The first category is the invalid pixel, whose corresponding depth z is zero. These pixels may represent noise or holes in depth images. Embedding features from invalid pixels may deliver improper information. The second category is the pixel, whose inverse project point is out of the neighbor ball of the query point. Points outside of the query point's neighbor ball may also be projected to pixels that are close to the projection of the query point, but these points are less related to the query point in 3D semantics, so they are also filter out. Finally, we gather the pixels within remaining pixels for fusion, whose inverse project points are in the K nearest neighbors of the query point.

For geometric-to-visual fusion in the l-th layer, given a query pixel, we first determine its inverse project point. Then the points which fall in the neighbor of the inverse project point with radius R^l_{g2v} are selected as candidate points. Finally, we gather the KNN points of the inverse project point within the candidate points.

C.3. Keypoint

In this work, we utilize dense descriptions for correspondence estimation. In other words, all the points in the point clouds are considered as keypoints except invalid points with depth z=0.

C.4. Geometric fitting

Our geometric fitting is the same as that in UR&R [14], which is a modified RANSAC [16]. We provide its detail for convenience in this section.

Given 400 input correspondences $C = \left\{\left(p^{\mathcal{S}}, p^{\mathcal{T}}, w\right)_i : 0 \le i < 2k\right\}$ with their corresponding weights, we randomly sample t subsets of C and estimate t candidate transformations. Each subset contains l randomly sampled correspondences, and each candidate transformation is estimated by solving a weighted Procrustes problem [37] on a subset. Then we choose the candidate transformation T^* with minimal error $E(C, T^*)$ in equation 7 as our final estimation. During training, we set t to 10 and l to 80. At test time, we set t to 100 and l to 20.

C.5. Differentiable rendering

The differentiable renderer is a rendering technique that leverages differentiable programming to optimize and compute gradients of the rendering process. Its basic principle is shown in Figure 5. It softens the process of projection, whereby each pixel is the accumulation of multiple splatted points. This allows each point to receive gradients

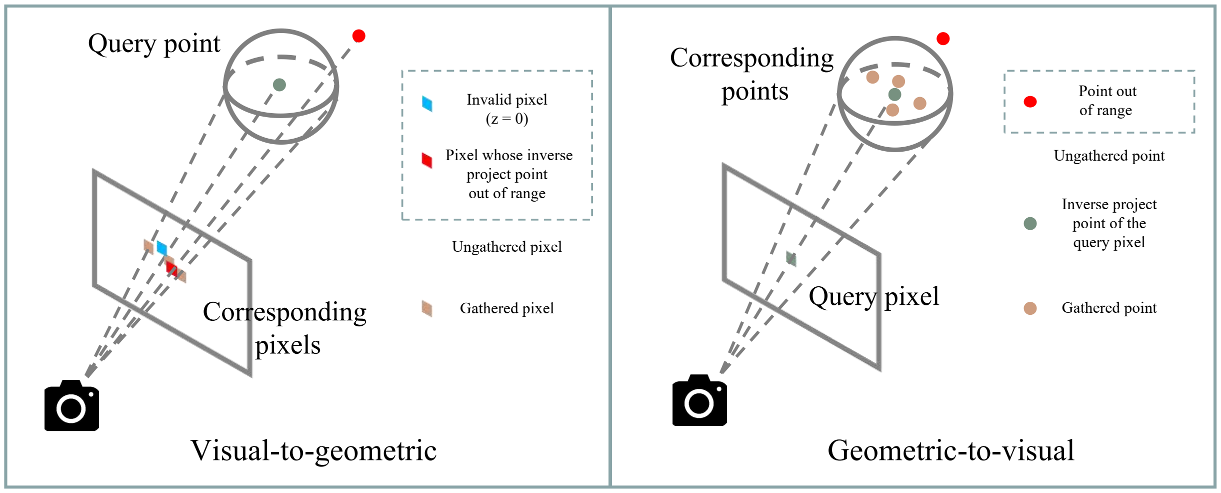


Figure 4. Point/pixel gathering for fusion.

from multiple pixels, avoiding the local gradient [33] issues caused by hard rasterization. Furthermore, the accumulation strategy employed in the soft projection approximates the occlusion observed in the natural world. We implement our differentiable renderer using Pytorch3d [53]. It takes transformed point clouds as input and outputs rendered images for photometric loss calculation.

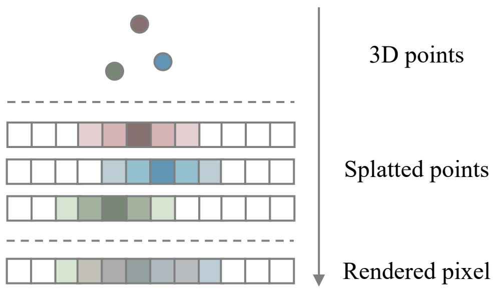


Figure 5. The basic principle of differentiable rendering. Differentiable rendering softens the process of projection. Each 3D point affects a certain region of pixels by splatting itself to a region, and the rendered pixels are the accumulation of all the splatted points.

D. Qualitative Visualization

We provide detailed visualization in this section. We visualize the inputs, extracted features, generated correspondences, and final registration results in two challenging scenes, including cluttered and ambiguous scenes.

The results of the cluttered scene are shown in Figure 7. The first two rows show the registration of two single branches, and the last row shows ours. It can be seen that in a cluttered scene, there exist complex semantics, partial overlap, and blur caused by camera jitter, which make registration challenging. Visual and geometric branches can not deal with it perfectly and tend to generate more outlier correspondences, leading to registration failure. But our method considers both semantics and local geometry, tends to avoid wrong correspondences.

The results of the ambiguous scene are shown in Figure 8. The first two rows show the registration of two single branches, and the last row shows ours. It can be seen that in an ambiguous scene, there exist many ambiguous and repetitive structures such as floors, walls, and symmetrical objects without textures, making correspondences based on a single modality contain a large proportion of outliers. Our fusion considers both semantics from RGB information and local geometric distributions from point clouds, which greatly improves the performance. For example, as shown in Figure 9, the visual features can not distinguish the hook from the armrest due to similar local texture and the geometric features produce more wrong correspondences because of too many repetitive surfaces in this scene. However, our fused features successfully distinguish them and produce correspondences from a more reliable area.

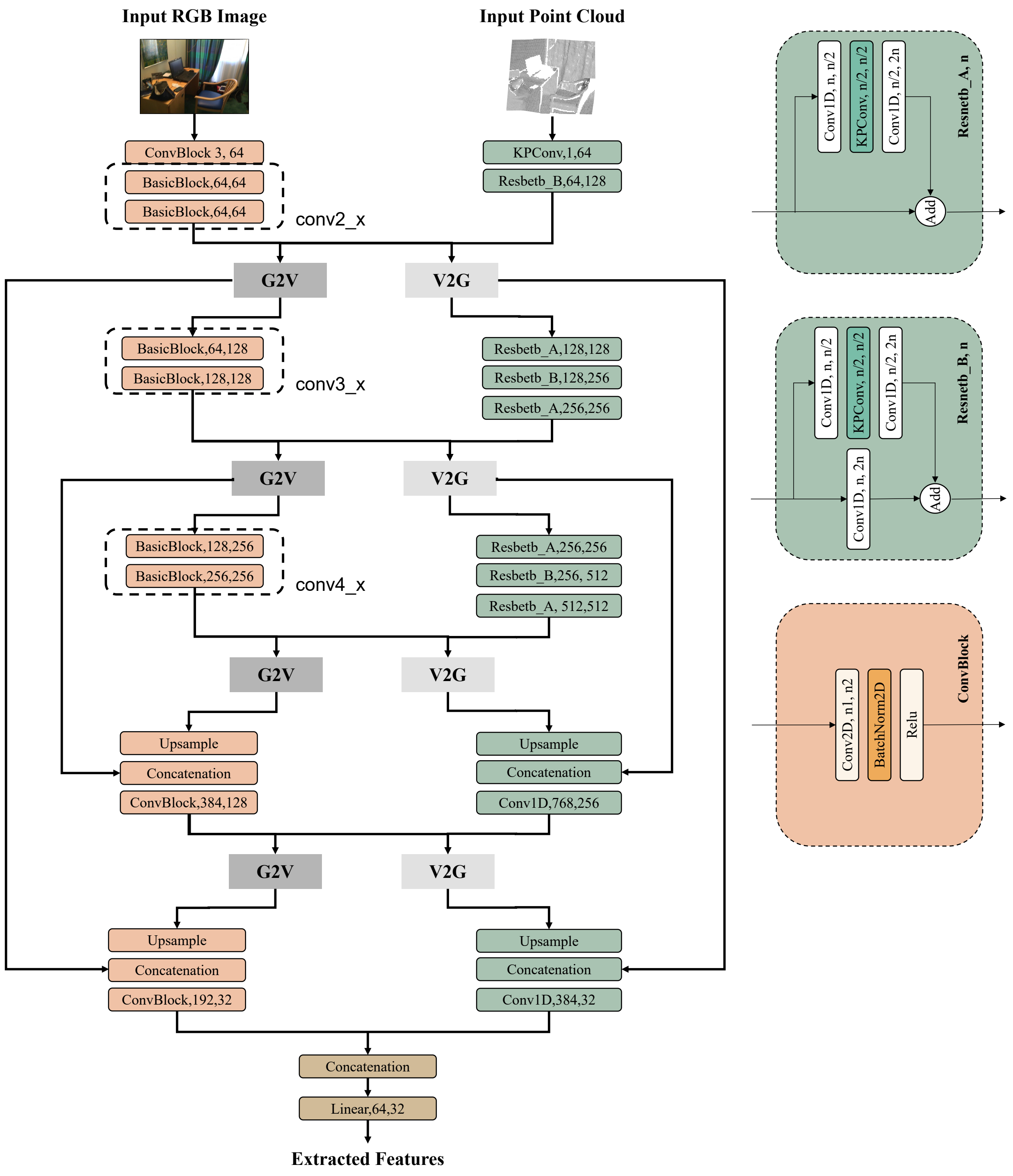


Figure 6. The detailed architecture of our feature extractor.

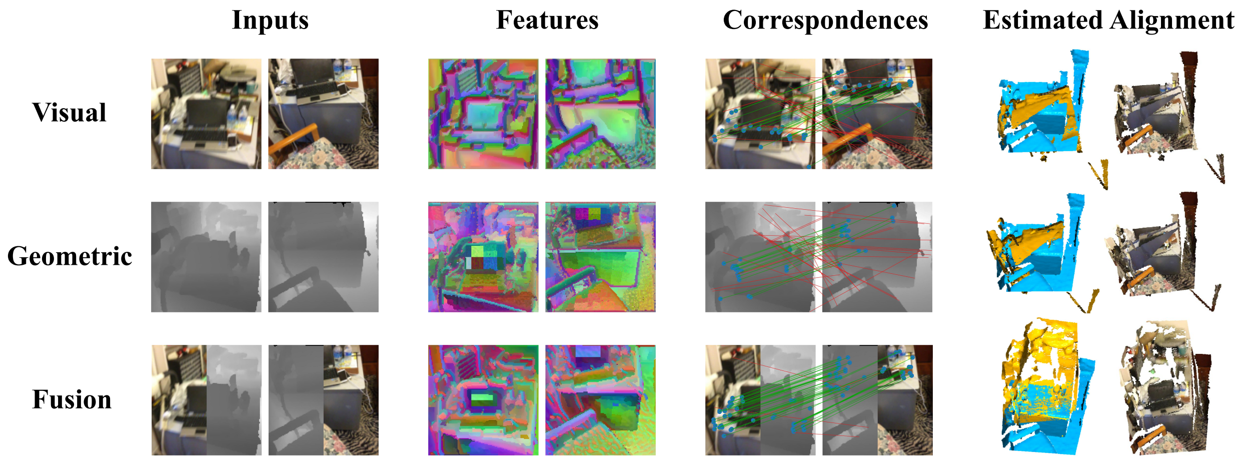


Figure 7. **Visualization on RGB-D registration in cluttered scene.** The features are visualized by mapping them to colors by t-SNE [58]. The red lines denote the outlier correspondences, while the green lines denote the inlier correspondences.

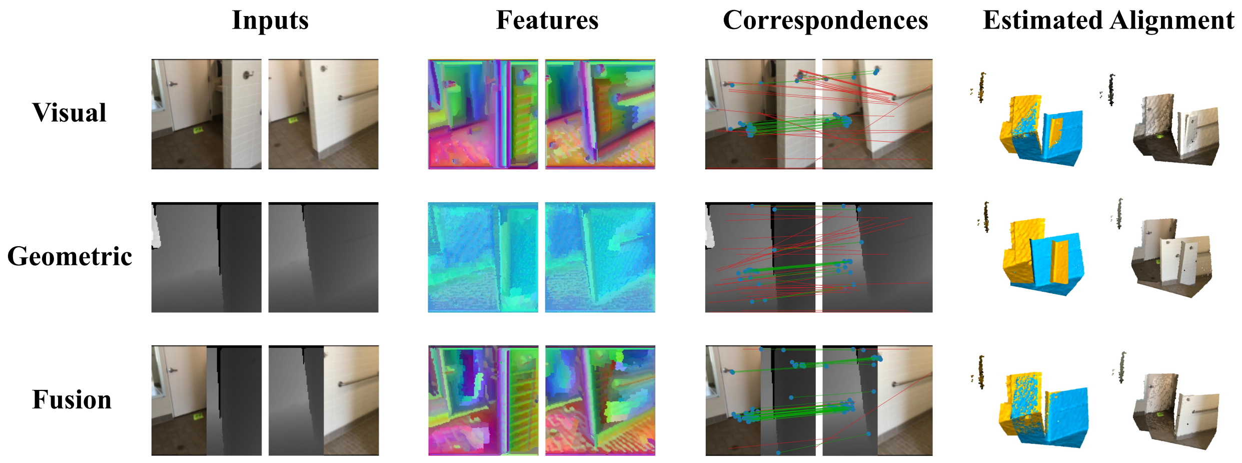
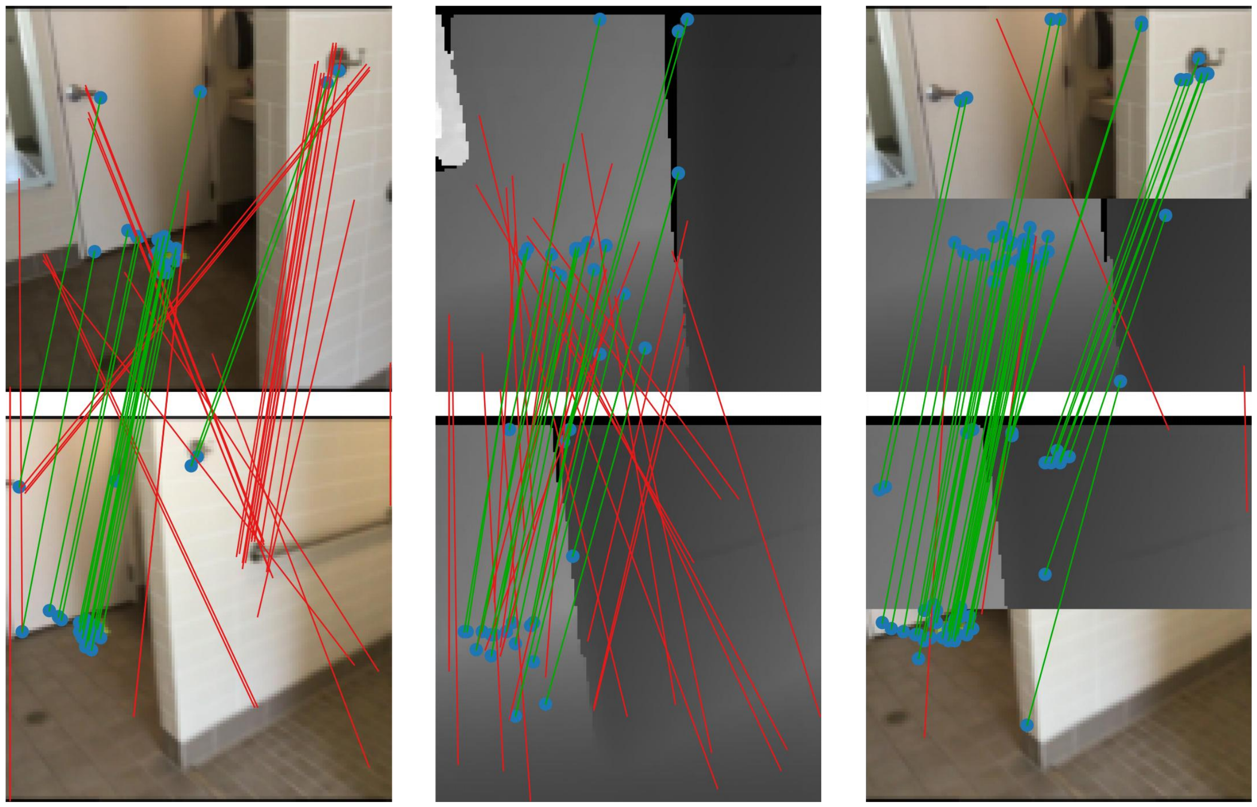


Figure 8. **Visualization on RGB-D registration in ambiguous scene.** The features are visualized by mapping them to colors by t-SNE [58]. The red lines denote the outlier correspondences, while the green lines denote the inlier correspondences.



(a) Correspondences from visual features

(b) Correspondences from geometric features

(c) Correspondences from our features

Figure 9. **Zoom-in visualization for correspondences in Figure 8.** The red lines denote the outlier correspondences, while the green lines denote the inlier correspondences. The visual features can not distinguish the hook from the armrest and the geometric features confuse the floor with the wall. But our fused features can generate reliable correspondences for registration as they consider the complementary information from RGB-D data.