Supplementary Material: Unsupervised Pointcloud Registration via Differentiable Rendering

Mohamed El Banani Luya Gao Justin Johnson
University of Michigan
{mbanani, mlgao, justincj}@umich.edu

A. Feature Encoder

We only have one learned component in our model, the feature encoder, which is implemented using the ResNet basic block defined in the `torchvision` library. Our feature encoder takes as input an RGB image. The first layer is a 2D convolution layer with a kernel size of 3 and and output channel dimension of 64. This is followed by two ResNet basic blocks that retain the spatial and feature dimensions of the activations. Finally, we use a 2D convolution layer to map the feature dimension from 64 to 32. We reduce the feature dimension to 32 as it allows us to use the fast kNN CUDA kernel defined in PyTorch3D [1]. All convolution layers are followed by BatchNorm and ReLU activation, except for the last layer.

B. Alignment Algorithms

During training, we use the Weighted Procrustes algorithm for point cloud registration since it allows us to maintain the differentiability of the pipeline. However, once the model is trained, we can use our pretrained features with other alignment methods such as RANSAC. Furthermore, we could use the Weighted Procrustes algorithm with other feature descriptors to better understand our model's performance. Hence, we evaluate the pairwise registration performance for different pairs of feature descriptors and alignment algorithms.

Given that the choice of correspondence set can be critical to the performance of an alignment algorithm, we compare two variants of RANSAC. In the first variant, we compute a large set of correspondences by simply finding the nearest neighbor to each feature. The second variant, RANSAC-Corr, filters the correspondences using Lowe's ratio test and only keeps the top 400. This is similar to our method and allows us to disentangle the impact of using Lowe's ratio to only filter the correspondences as opposed to using it to both filter and weigh the correspondence, as done by the Weighted Procrustes algorithm. For both variants, we use the implementation provided by Open3D with a limit of 100,000 iterations for RANSAC.

		Rotation		Translation		Chamfer	
Features	Estimator	Mean	Med.	Mean	Med.	Mean	Med.
SIFT	RANSAC	25.4	5.4	33.8	12.5	54.3	2.5
SuperPoint	RANSAC	13.1	4.0	20.9	10.6	30.0	1.4
FCGF	RANSAC	10.9	4.9	20.3	11.8	15.9	2.2
Ours	RANSAC	10.6	4.6	19.9	11.0	9.9	1.6
SIFT	RANSAC-Corr	18.6	4.3	26.5	11.2	42.6	1.7
SuperPoint	RANSAC-Corr	8.9	3.6	16.1	9.7	19.2	1.2
FCGF	RANSAC-Corr	9.5	3.3	23.6	8.3	24.4	0.9
Ours	RANSAC-Corr	3.5	1.8	8.5	5.6	4.4	0.5
SIFT	Ours	14.5	2.0	26.5	5.7	20.8	0.5
SuperPoint	Ours	4.8	1.6	8.5	4.1	7.4	0.3
FCGF	Ours	15.3	4.3	34.8	11.6	28.9	2.0
Ours	Ours	3.4	0.8	7.3	2.3	5.9	0.1

Table 1. **Impact of feature and alignment algorithm on Pairwise Registration.** Using Lowe's Ratio test to filter correspondences improves performance as expected, while using it to further weight the correspondences in the Weighted Procrustes algorithm further improves performance. Furthermore, our features outperform the baselines regardless of choice of alignment algorithm.

We present the results in Table 1. We first observe that filtering the correspondences improves the performance of all feature descriptors as shown by the improved performance of RANSAC-Corr and Weighted Procrustes. We note that some of those differences would decrease by allowing RANSAC to run for a longer number of iterations. However, even with 100,000 iterations, we find the inference with RANSAC can be up to 10x slower than using Weighted Procrustes. Intuitively, filtering correspondences should improve the performance since it makes it easier for RANSAC to find the best alignment by decreasing the number of outliers. This is especially true for methods that output a large number of feature descriptors like FCGF and our method. Finally, we observe that using Weighted Procrustes further improves the performance for all visual features.

C. Qualitative Results

We include additional qualitative results in the appendix to provide a clear picture of our model's estimates as well

as its limitations. First, we show the additional registration results in Figure C.1. Our model can extract dense correspondences between images, allowing it to accurately estimate the camera motion in the scene and register the images. Similar to registration methods, our approach struggles when there is limited visual and geometric texture in the image. This results in the failure mode shown in the bottom row of Figure C.1. However, as shown in the second to last row, the dense correspondences can still help it identify the correct correspondences.

Second, we also show the RGB-D renderings produced by the model in Figure C.2. As can be seen, our model's reconstruction from the rendered features closely matches the appearance of the input images. Furthermore, as discussed in Section 3.4, our model only renders the overlapping sections of the scene, resulting in the sections of the image not being rendered. As presented in the image, our image pairs do not have significant overlap, demonstrating our model's ability to perform wide-baseline correspondence matching and point cloud registration.

References

[1] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv*, 2020.

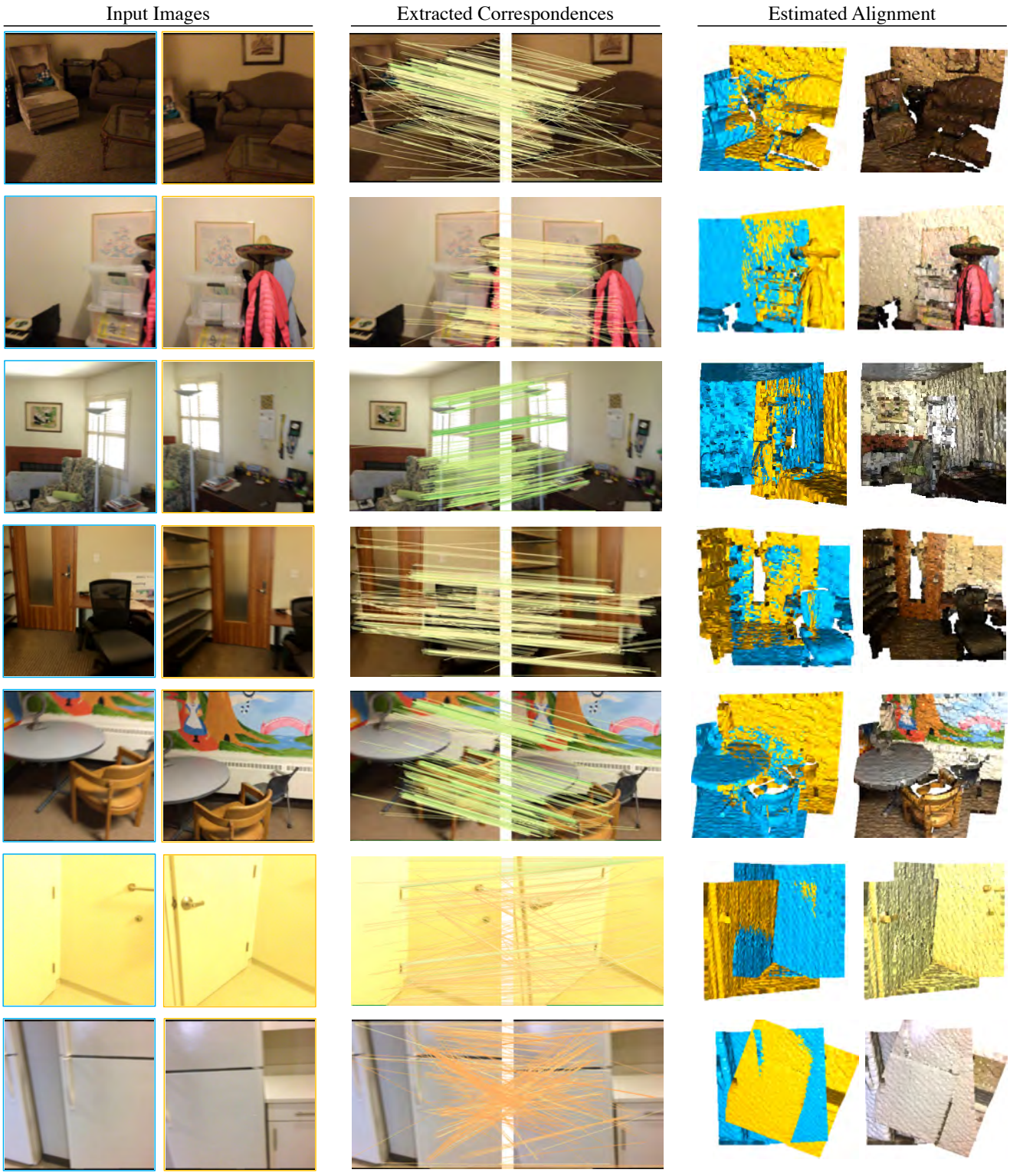

Figure C.1. **Pairwise Registration Results.** Our model can extract dense correspondences across different scenes. Our model is also capable of accurately registering the scene despite the existence of outliers. Finally, while our model struggles with finding correspondences in limited visual texture, it can still accurately register the scene in some cases. In more difficult cases, the estimated correspondence weights are much lower, as indicated by the orange color.

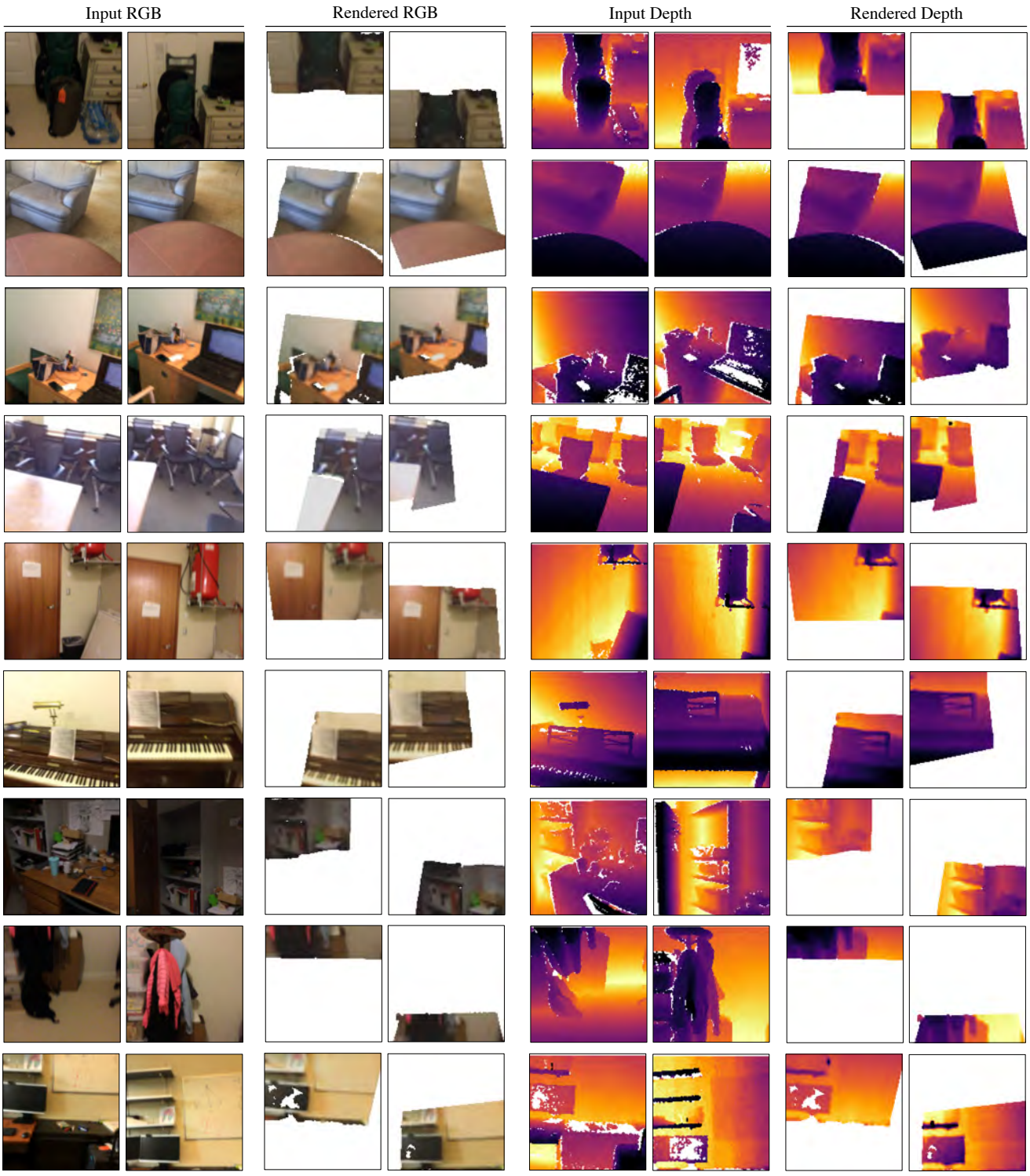

Figure C.2. **RGB-D Rendering Results.** Our model accurately aligns and renders the scene. While missing depth pixels in the input are due to sensor issue, missing pixels in our renders can be due to non-mutually visible surfaces either due to change in camera view, occlusion, or missing depth.