Fine-To-Coarse Global Registration of RGB-D Scans Supplemental Material

Maciej Halber
Princeton University
mhalber@cs.princeton.edu

Thomas Funkhouser
Princeton University
funk@cs.princeton.edu

1. Algorithm Details

This section provides supplemental technical and implementation details in support of Section 4 of the paper.

1.1. Preprocessing

Extracting Features As described in the main paper, for each input image $I[k]$ we extract a set of features $F[k]$. In our implementation we use 4 feature types:

1. **Plane** All parts of the image that are locally planar, i.e. can be represented as a position and a normal. We construct a planar surface feature for each pixel in coplanar clusters of size ≥ 100.
2. **Silhouette** We have found it useful to mark depth discontinuities with a separate feature type. These are represented as a point, normal and a direction along the silhouette. Normal and direction are estimated with PCA on neighboring silhouette pixels. The same approach is used to estimate direction for **Ridge** and **Valley** features.
3. **Ridge** If two large planes are intersecting and the angle between their normals is larger than π we mark the intersection as a ridge feature.
4. **Valley** - If two large planes are intersecting and the angle between their normals is smaller than π we mark them as a valley feature. Both valley and ridge features are represented as a position, normal (average of the normals of neighboring planes) and direction along the intersection.

We subsample the initial set of features using the Poisson Dart Algorithm, with a minimum spacing between the features equal to $0.05m$. We first visit all the silhouette edges, then the ridges and valleys, followed by the planar features. This process allows us to create a uniformly sampled set of features for each image $I[k]$, which improves the closest point constraint search times.

In our implementation we used a different weights to indicate the importance and our confidence for each feature type. Values used were $\omega_{plane} = 1.0$, $\omega_{silhouette} = 0.6$, $\omega_{ridge} = 0.2$, $\omega_{valley} = 0.2$.

Creating Base Planar Proxies To detect a set of planar proxies B in images I we have implemented our own agglomerative hierarchical clustering algorithm. We have found it to perform better that region growing or RANSAC algorithms on a noisy dataset like SUN3D. The low-level algorithmic details are:

- Apply a bilateral filter to reduce noise and quantization effects ($\sigma_{xy} = 3px$, $\sigma_{depth} = 5cm$).
- Mark pixels as boundaries if their depths differ from any of their neighbors by more than 10%.
- Compute connected components by partitioning the image on boundaries
- Estimate normals for pixels using RANSAC on neighborhoods of radius $r = 10cm$ within the same connected component.
- Compute sets of coplanar pixels using hierarchical clustering.
- Refine clusters with a RANSAC algorithm to reassign pixels to their largest compatible cluster.
- For each cluster insert proxy to B. Assign the centroid, normal and radius for the proxy based on PCA of the associated pixels.

Each resulting planar proxy is represented by a centroid and a normal $B[j] = \{p, \vec{n}\}$.

Aligning Adjacent Images The frame-to-frame tracking that is used in our implementation is based on system proposed int Xiao et al.[10]. We begin by extracting matching SIFT features in consecutive RGB images. This process returns two sets of image locations $\mathcal{P'}_k$ and $\mathcal{P'}_{k+1}$. For each set of image locations we reject candidates that do not have valid depth value in the accompanying depth image, to create pruned out sets $\mathcal{P}_k$ and $\mathcal{P}_{k+1}$. Since all points in both sets now

have valid depths, they can be back-projected into 3D using the intrinsics matrix K, to create $\mathcal{X}_k$ and $\mathcal{X}_{k+1}$. We use RANSAC to find a transformation that matches most of the back-projected points in order to create a pairwise transformation $L[k]$.

Initializing Transformations We then concatenate the transformations to create an initial set of poses for our optimization T_0.

$$(T_0[0] = I; T_0[j] = L[j]T0[j-1]; j \in [1; k])$$

1.2. Fine-to-Coarse Refinement

Creating Co-planarity Constraints. When clustering coplanar proxies B we again use an agglomerative clustering algorithm that merges the transformed proxies based on their pairwise similarity. Similarity is expressed as a product of pairwise point-to-plane distance factors f_{pp} and normal deviation factors f_{nd}. For pair of proxies B_a, B_b:

$$S_{ab} = f_{pp}(B_a, B_b) f_{pp}(B_b, B_a) f_{nd}(B_a, B_b)$$

Factors are expressed with respect to set maximum thresholds. In our implementation the max pairwise distance was set to $t_d = 30cm$ and the maximum normal deviation to $t_a = \frac{\pi}{8}$

$$f_{pp}(B_a, B_b) = \begin{cases} 1.0 - \frac{d(p_a, B_b)}{t_d} & \text{if } d(p_a, B_b) \geq t_d \\ 0.0 & \text{if } d(p_a, B_b) < t_d \end{cases}$$

$$f_{nd}(B_a, B_b) = \begin{cases} 1.0 - \frac{\angle(\vec{n_a}, \vec{n_b})}{t_a} & \text{if } \angle(\vec{n_a}, \vec{n_b}) \geq t_a \\ 0.0 & \text{if } \angle(\vec{n_a}, \vec{n_b}) < t_a \end{cases}$$

Result of this procedure constitutes set of cluster proxies P

Creating Planar Relationship Constraints. When deciding on the type of the relationship and its weight we use exponential function with $\sigma_\theta = 0.13$. This value will assign negligible weights to relationships that deviate more that 15° from the desired angle. It is possible that this parameter should be adjusted iteratively, but we have not spent much time investigating or tweaking it.

Creating Feature Correspondence Constraints. For finding closest point correspondences we used the initial thresholds for distance and normal deviation were

$$\{d_{initial} = 0.5m, \alpha_{initial} = 30^\circ\}$$

and for final thresholds:

$$\{d_{final} = 0.2m, \alpha_{final} = 20^\circ\}$$

These values are adjusted based on the distance alongside trajectory between images whose features are currently considered. The initial values decrease as a square root with this distance, where the maximum is reached when the distance is equal to $0.5l_i$ and the minimum is when the distance is 0;

Although the error function presented in the paper is non-linear, we find that our overall algorithm converges more quickly when the variables are relaxed with a linear approximation once per iteration. To solve the system of linear equations we used the CSparse solver [4]. Though this approach is related to solving the non-linear function with a series of linear approximations (e.g., as is common in solvers like Ceres [1]), there is a significant difference — we update the structure of the error function between solving every linear approximation (by finding new structural and closest point constraints). This is beneficial, since solution to the updated function is generally closer to the final solution.

1.3. Optimization

For all our experiments we have run our algorithm using the same set of parameters:

- Starting window length $l_i = 3m$.
- Number of iterations $n_{iter} = 12$.
- Weights for energy function terms:
  - Initial: $w_C = 1500$, $w_L = 1000$,, $w_H = 1500$, $w_G = 1500$
  - Final: $w_C = 1000$, $w_L = 1000$, $w_H = 1000$, $w_G = 1000$

Similarly to the feature correspondence thresholds, the weights are linearly interpolated between initial and final values throughout the optimization.

Planar Structure Error. The coplanar misalignment E_{cp} of one cluster proxy P_a to P_b is computed by summing the squared distances from samples of points q_s ($s \in [1, s_{max}]$). Each q_s is either sampled from the boundary of a 1 meter radius disk around p_a, or is at the same location as p_a (In our experiments $s_{max} = 5$):

$$E^{\rightarrow}_{cp}(A, B) = \sum_{s=1}^{s_{max}} ((q_A - q_B) \cdot n_B)^2$$

$$E_{cp}(A, B) = E^{\rightarrow}_{cp}(A, B) + E^{\rightarrow}_{cp}(B, A)$$

Local Alignment Error. We compute the misalignment E_t of one rigid transformation T_j to another T_k in the neighborhood of a point c_j by summing the squared distances between points p_s ($s \in [1, s_{max}]$)

sampled uniformly on a 1 meter radius sphere when they are transformed by T_j versus T_k ($s_{max} = 8$):

$$E_t(T_j, T_k) = \sum_{s=1}^{s_{max}} (T_j(p_s) - T_k(p_s))^2$$

These formulations provide error terms measured in squared distances between corresponding points (rather than differences of matrix elements, angles, viewpoints, etc.) and thus are more natural to combine with other error terms of the same form in our multi-objective optimization (as noted in [7]).

2. Ground Truth Correspondence Collection

To collect the set of correspondences required for our evaluation metric we have designed a user interface to assist in the collection process, as well as to evaluate the quality of correspondences. The interface can be seen in figure 1. The user is presented with view of image-pointcloud pairs. The user can freely scroll through the sequence to select corresponding pairs. Once a pair of images is marked the user can begin clicking points. Anytime a point is clicked the 3D view is also updated, which gives the user a sense where exactly a point lands in 3D space.

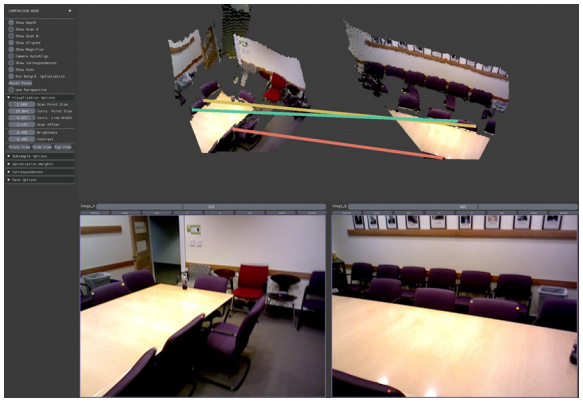

Figure 1: Interface for collecting ground truth correspondences

Our interface also has an evaluation mode, which runs an iterative reconstruction algorithm with pairwise transformations and ground truth correspondences as constraints (Figure 2). We found this to be an easy way to validate that the clicked correspondences are correct, as an erroneous ground correspondence will lead to big errors in the resulting reconstruction and can be easily fixed by the user.

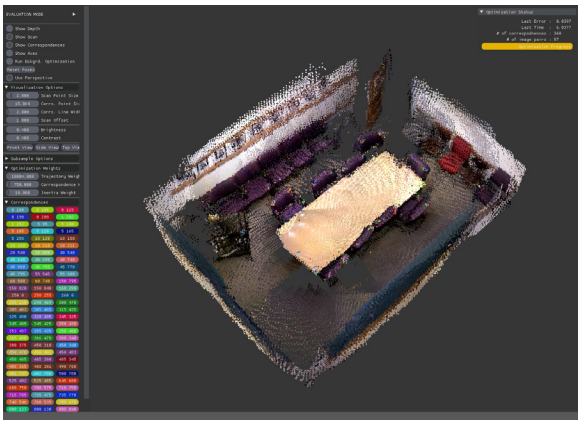

Figure 2: Interface showing resulting reconstruction. On the side panel you can see colored button relating to pairs of images marked by the user.

3. Quantitative Results

Below we present the full set of results we used to produce a Table 1 in the paper. We also add results produced with two realtime tracking methods, Elastic Fusion [9] and Kintinuous [8]. Note that real-time methods are not expected to perform as well, due to the trade-offs one needs to make for a real-time system.

We would also like to note that Choi et al.[2] does not perform well on many sequences in our test set, as these sequences often are lacking strong geometrical features, and most of the tracking depends on RGB data. Since Robust Reconstruction [2] does not use the color channel, we do not expect it to be able to deal with such cases.

3.1. Robust Reconstruction

We have run the Robust Reconstruction[2] system with the default settings for SUN3D dataset, published by the authors in their downloadable binary package. The only change made was to modify the *FragmentOptimizer.exe* parameter from *–slac* to *–rigid*, which led to better results on SUN3D dataset. In running their code, we obtained slightly different results than those published in their paper. Though the difference is minor, we present both results in Table 2 to avoid confusion in other comparisons.

4. Qualititative Results

To demonstrate the robustness of our algorithm we show the results on the SceneNN [5], BundleFusion [3] and SUN3D [10] datasets.

SceneNN We show the resulting reconstructions of 6 SceneNN scenes (Figure 3), produced using VoxelHashing [6] with trajectories pre-computed by our algorithm. We also show the overhead pointcloud ren-

Sequence Name	Ours	SUN3D	RR	Elastic Fusion	Kintinuous
brown_bm_1	0.08345	0.25424	1.60400	1.90877	1.15671
brown_bm_4	0.10545	2.00690	4.12032	0.64936	1.78738
brown_cogsci_1	0.07161	0.89468	1.52869	0.75887	0.55985
brown_cs2	0.06346	0.21408	3.55556	0.89136	0.47414
brown_cs3	0.10796	1.90186	5.90101	2.90157	1.58114
hv_c11_2	0.06471	0.40341	0.27989	0.18390	0.15577
hv_c3_1	0.06541	0.09465	0.41692	0.30158	0.31309
hv_c5_1	0.07766	0.26991	0.11158	0.29152	0.28333
hv_c6_1	0.07524	0.62119	0.26693	0.27570	0.30313
hv_c8_3	0.08656	0.45715	0.24724	0.38132	0.28994
home_at_scan1_2013_jan_1	0.04063	0.21196	0.07570	1.18692	1.23930
home_bksh_oct_30_2012	0.05871	0.15002	1.23549	1.47723	0.58745
home_md_scan9	0.06063	0.16358	1.04740	1.29805	0.54559
nips_4	0.05109	0.15168	0.06181	0.45188	0.40953
scan1	0.06788	0.52143	1.91663	1.98147	1.46379
scan3	0.05042	0.07849	0.06207	0.13804	0.13694
maryland_hotel1	0.06140	0.30138	0.05156	0.65117	0.25950
maryland_hotel3	0.05794	0.20083	0.05260	0.15046	0.11797
d507_2	0.13874	0.32074	0.08354	0.57447	0.52683
ted_lab_2	0.04699	0.11556	0.05600	0.61538	0.59755
76-417b	0.04852	0.09020	0.04724	0.70408	0.68069
76-1studyroom2	0.05347	0.17491	0.12469	0.55497	0.27545
dorm_next_sj	0.08861	0.21222	0.23403	0.19009	0.12923
lab_hj	0.09000	0.67366	0.10347	0.47529	0.16703
sc_athena	0.09680	0.13690	1.41592	1.40803	0.23490

Table 1: Detailed quantitative results. RR abbreviates Robust Reconstruction by Choi et al.[2]. Errors are RMSE of distances of corresponding points given the trajectory produced by each method, in meters.

Sequence Name	Ours	RR	RR(published)
hv_c5_1	0.0689	0.1116	0.1938
hv_c6_1	0.0680	0.2669	0.2403
hv_c8_3	0.0731	0.2472	0.2007
maryland_hotel3	0.0606	0.0526	0.567
d507_2	0.1196	0.0835	0.2506
76-1studyroom2	0.0567	0.1247	0.2497
dorm_next_sj	0.0905	0.2340	0.2583
lab_hj	0.0898	0.1035	0.1045

Table 2: Result comparison between trajectories computed with Choi et al. [2](abbreviated as RR), and trajectories published at http://redwood-data.org/indoor/models.html. Errors are RMSE of distances of corresponding points given the trajectory produced by each method, in meters.

derings of other reconstructed scenes in figure (Figure 14). Note that we provide the qualitative results, as there is no ground truth or quantitative metric to evaluate them.

BundleFusion We also show results of reconstructions of the 8 scenes published recently with BundleFusion [3](Figure 4). Our reconstructions match the quality of the results presented on the BundleFusion website[1], and are created with same parameters we used for our SUN3D evaluation. Again we show the qualitative results as there is neither ground truth information or code released for BundleFusion (as of writing of this document).

SUN3D Lastly we also present visualizations of 374 scenes reconstructed from the SUN3D dataset. We found that our method produces good results in 350 cases (Figures 5,6,7,8,9,10,11,12). For 16 cases the resulting reconstructions have errors, mostly due to lost tracking by SUN3DSfM during the initialization

[1] http://graphics.stanford.edu/projects/bundlefusion/

phase which leads to separate parts of the reconstruction that are aligned well, but that are not aligned with each other (Figure 13). For 8 cases we have observed some unexpected aberrations, most likely due to issues with pairwise matching code producing initializations.

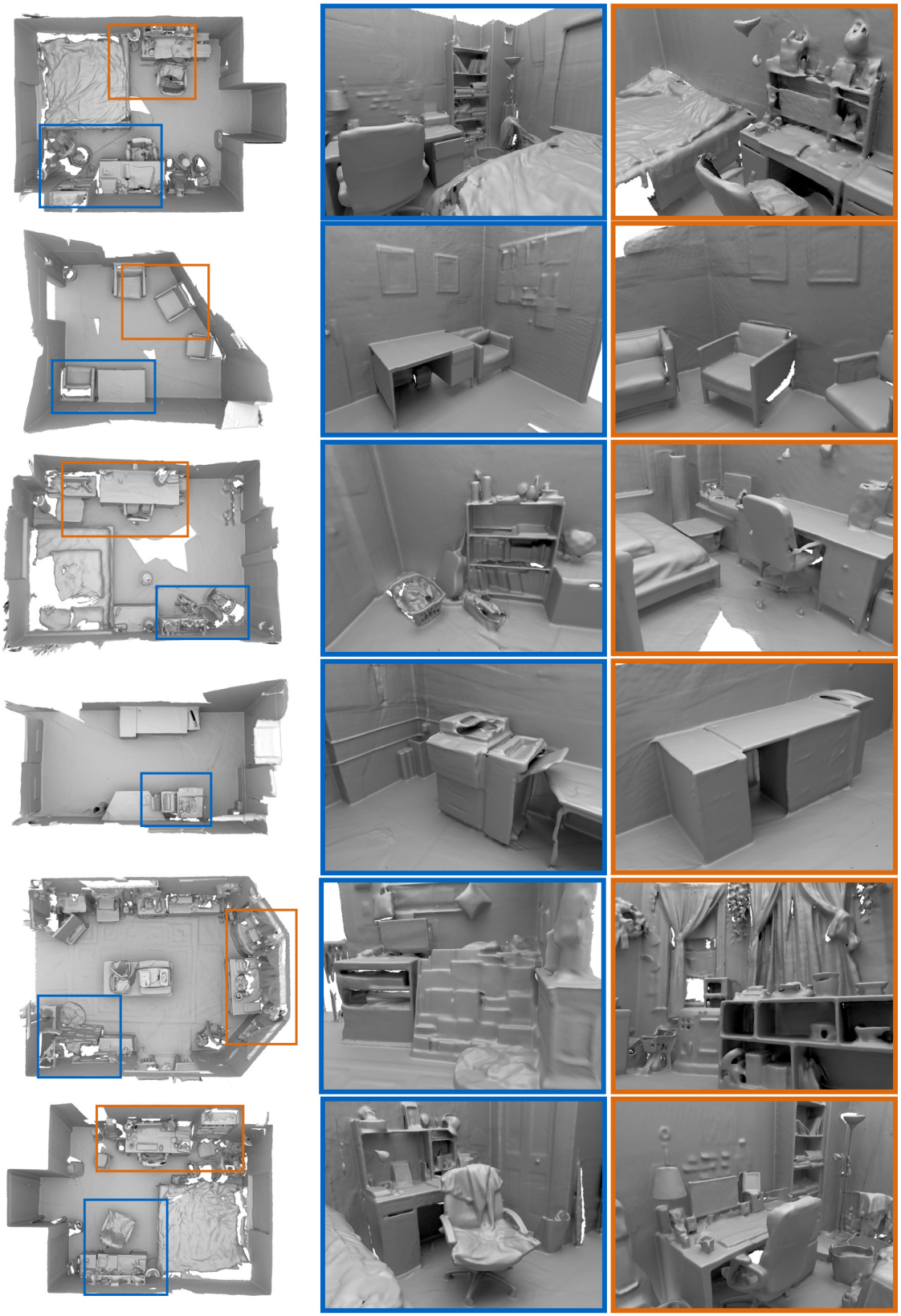

Figure 3: High quality renderings of SceneNN scenes. On the right hand side we show birds-eye view of the scene with regions marked, locating zoom-in views. Note that our alignment is able to produce reconstructions that both preserve the overall structure of the room, and low-scale geometric details.

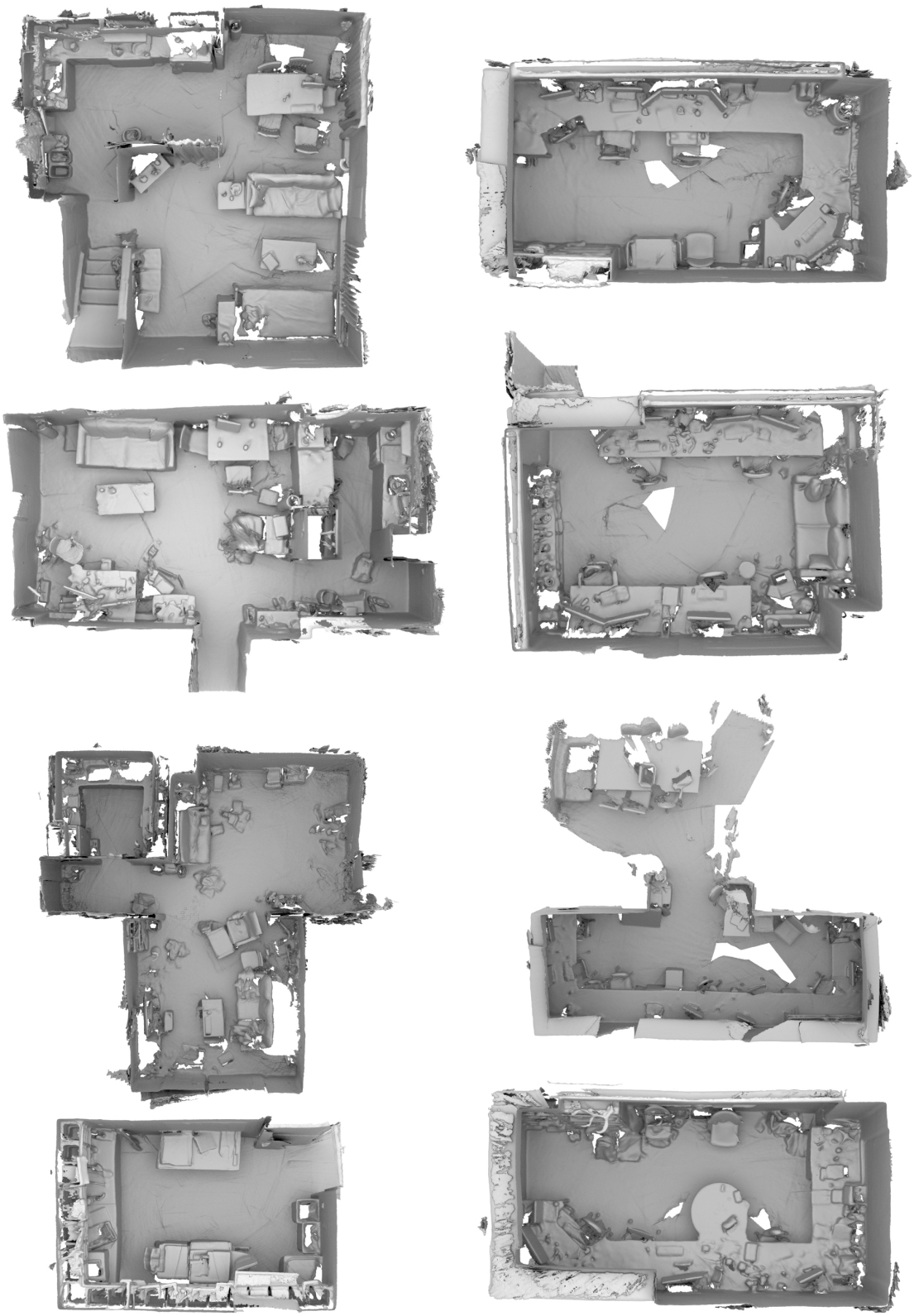

Figure 4: Overhead renderings of BundleFusion scenes. Note how our results preserve underlying structure of the rooms, keeping the walls straight, while also producing high quality local alignments, comparable to those in [3]

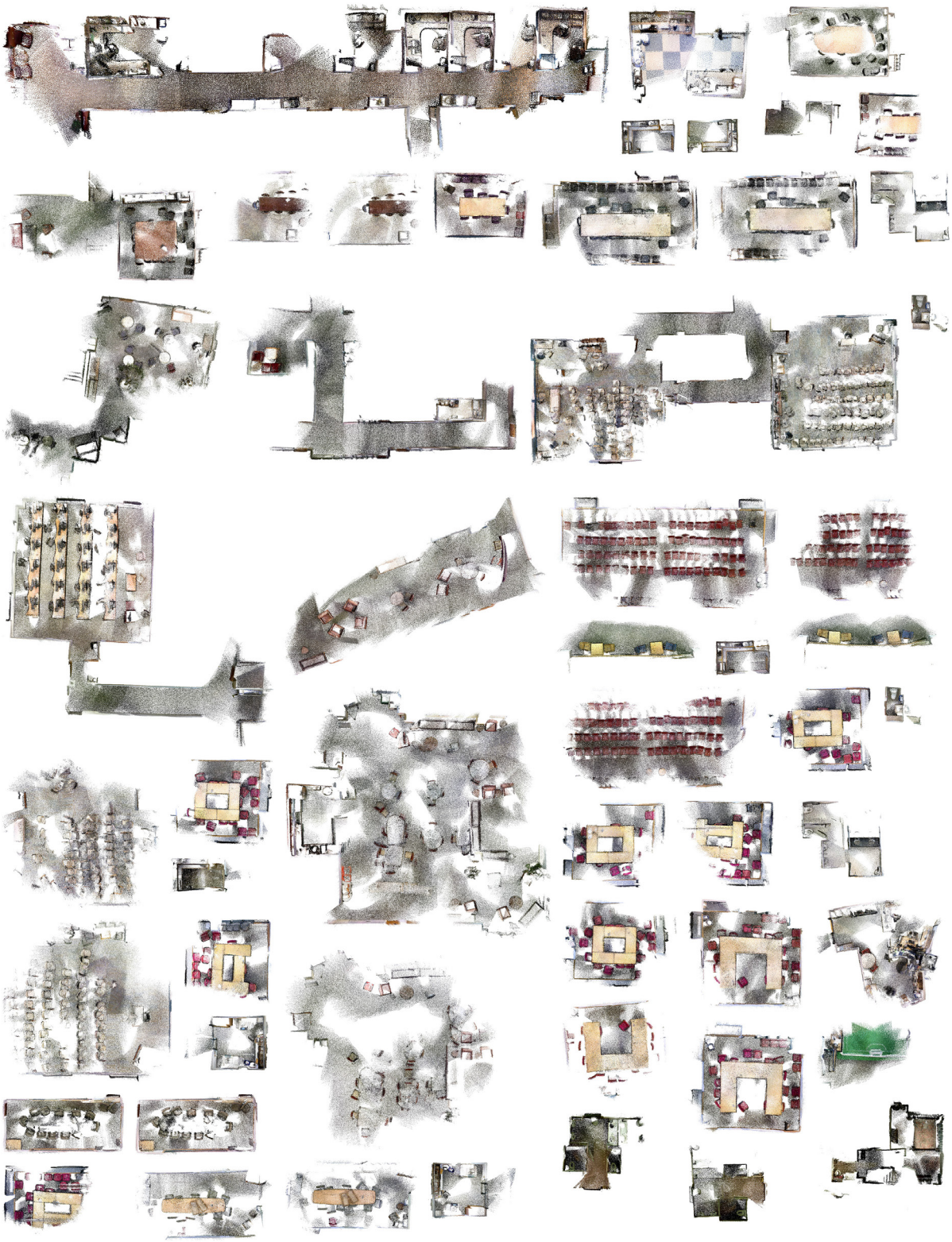

Figure 5: Overhead pointcloud renderings of SUN3D scenes

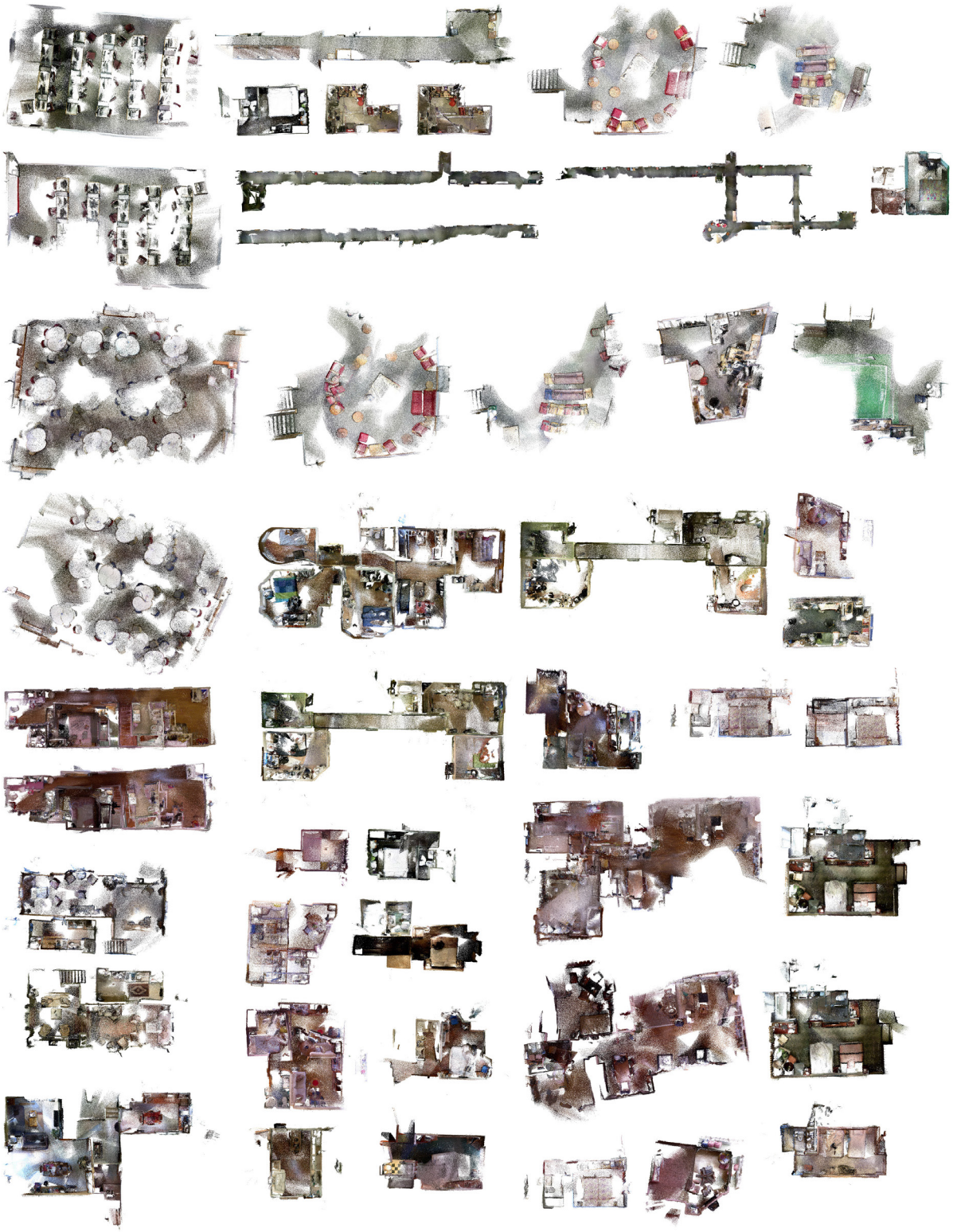

Figure 6: Overhead pointcloud renderings of SUN3D scenes

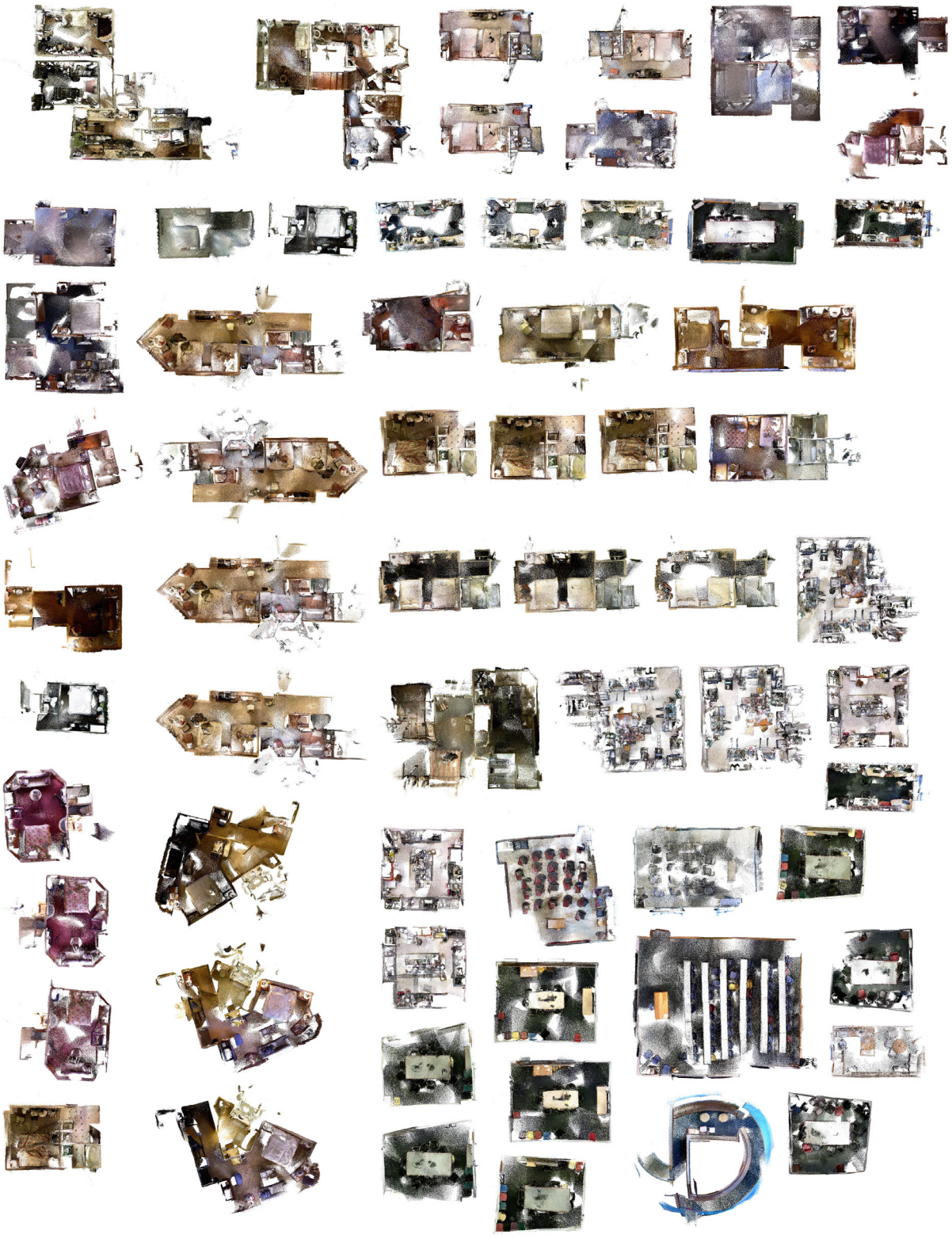

Figure 7: Overhead pointcloud renderings of SUN3D scenes

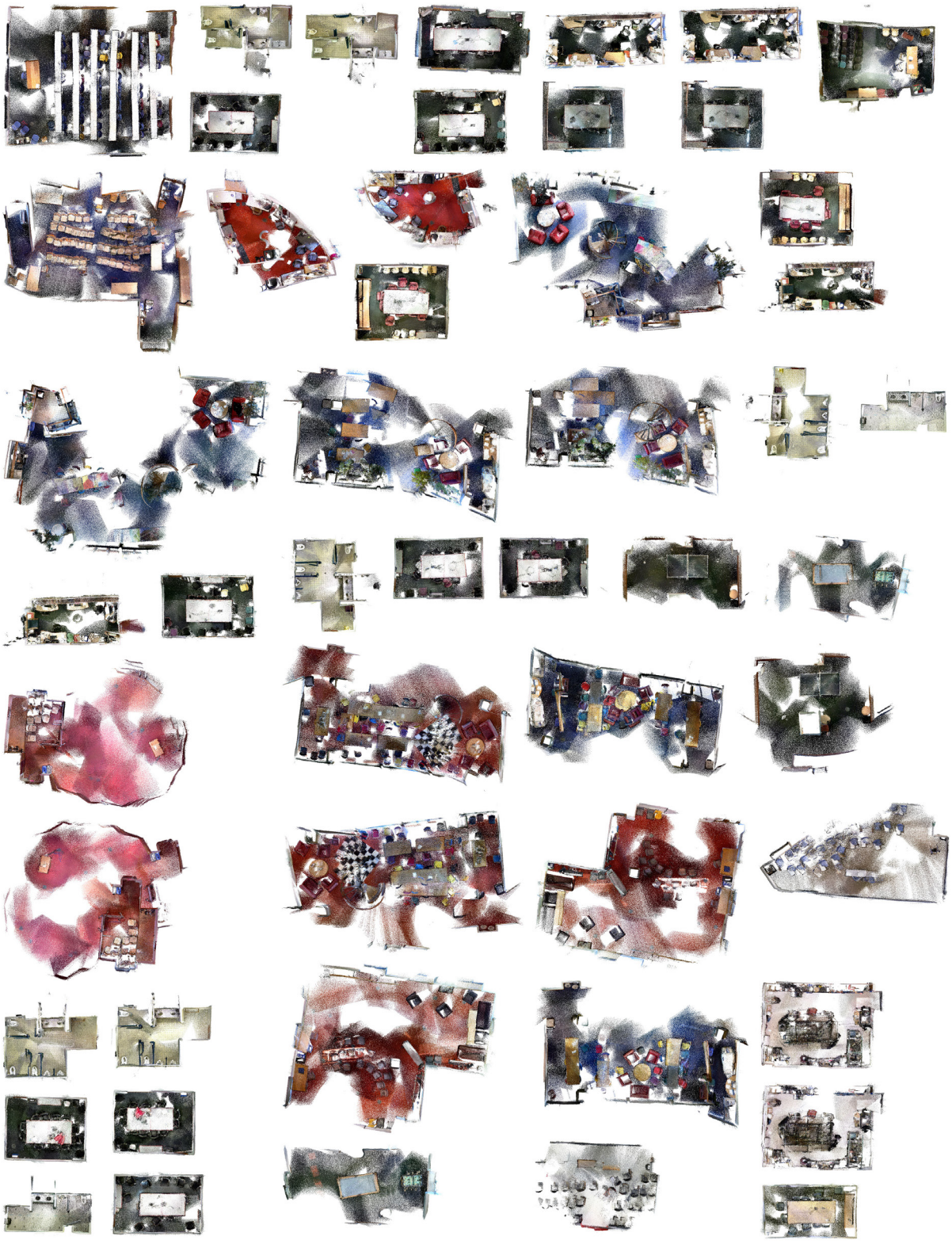

Figure 8: Overhead pointcloud renderings of SUN3D scenes

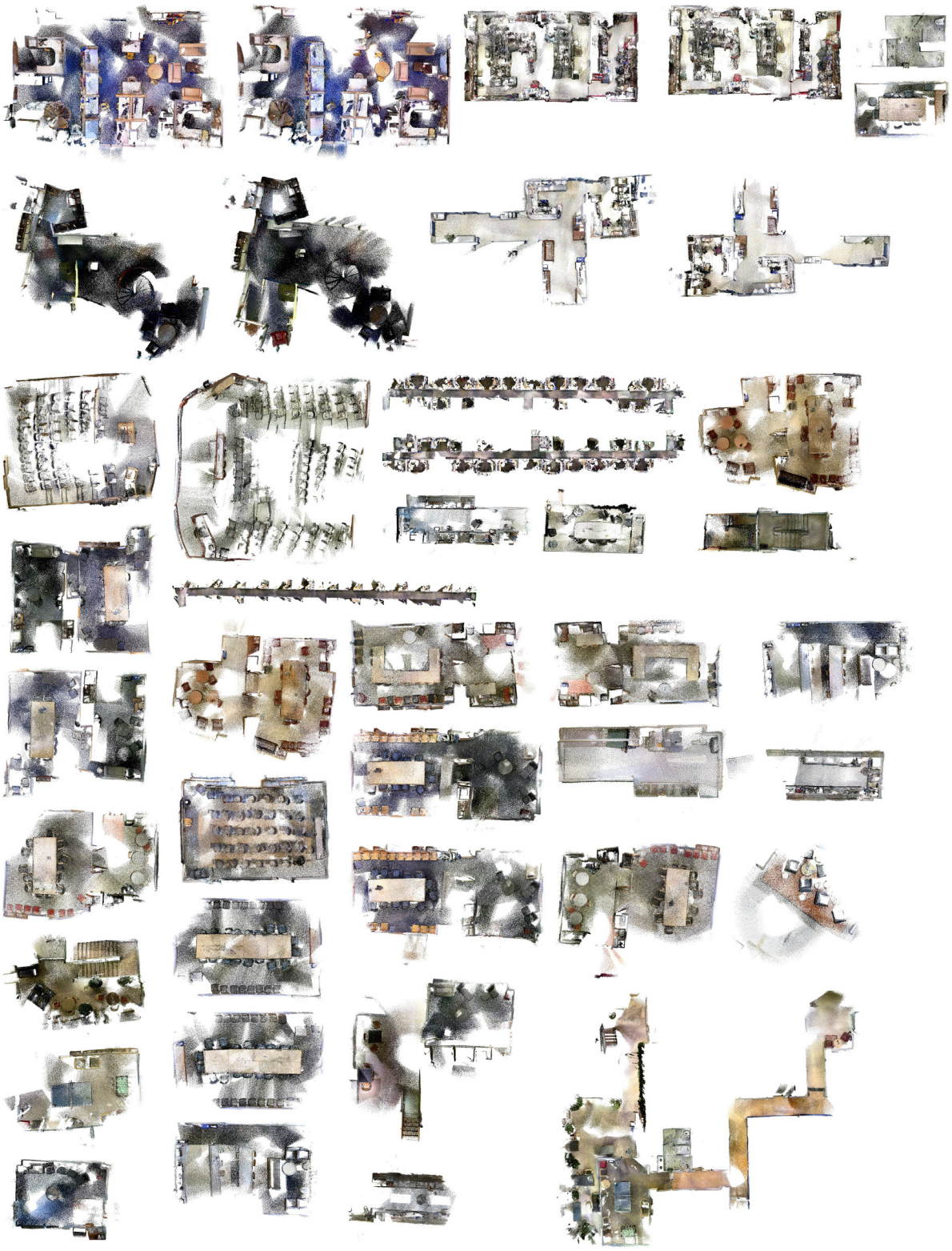

Figure 9: Overhead pointcloud renderings of SUN3D scenes

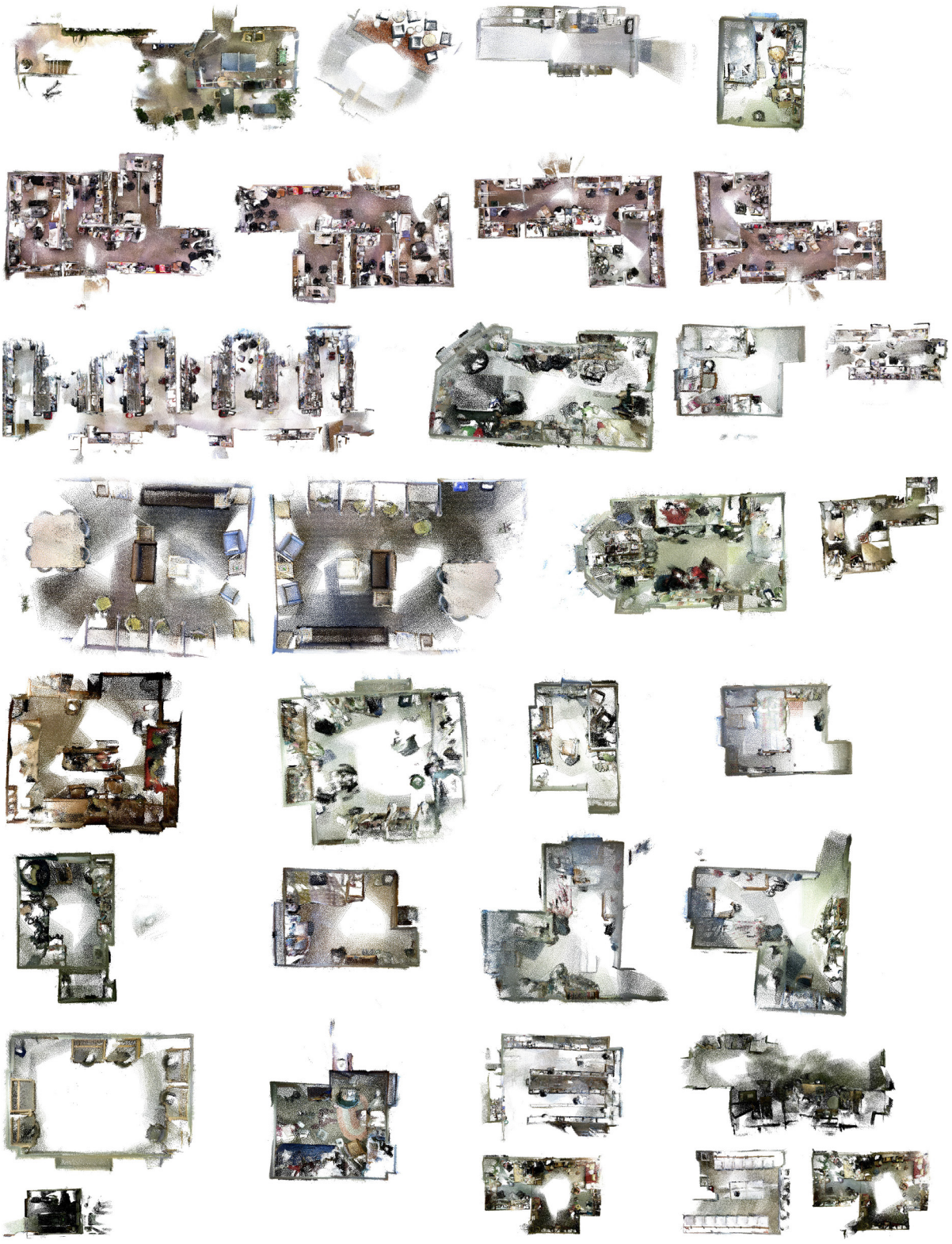

Figure 10: Overhead pointcloud renderings of SUN3D scenes

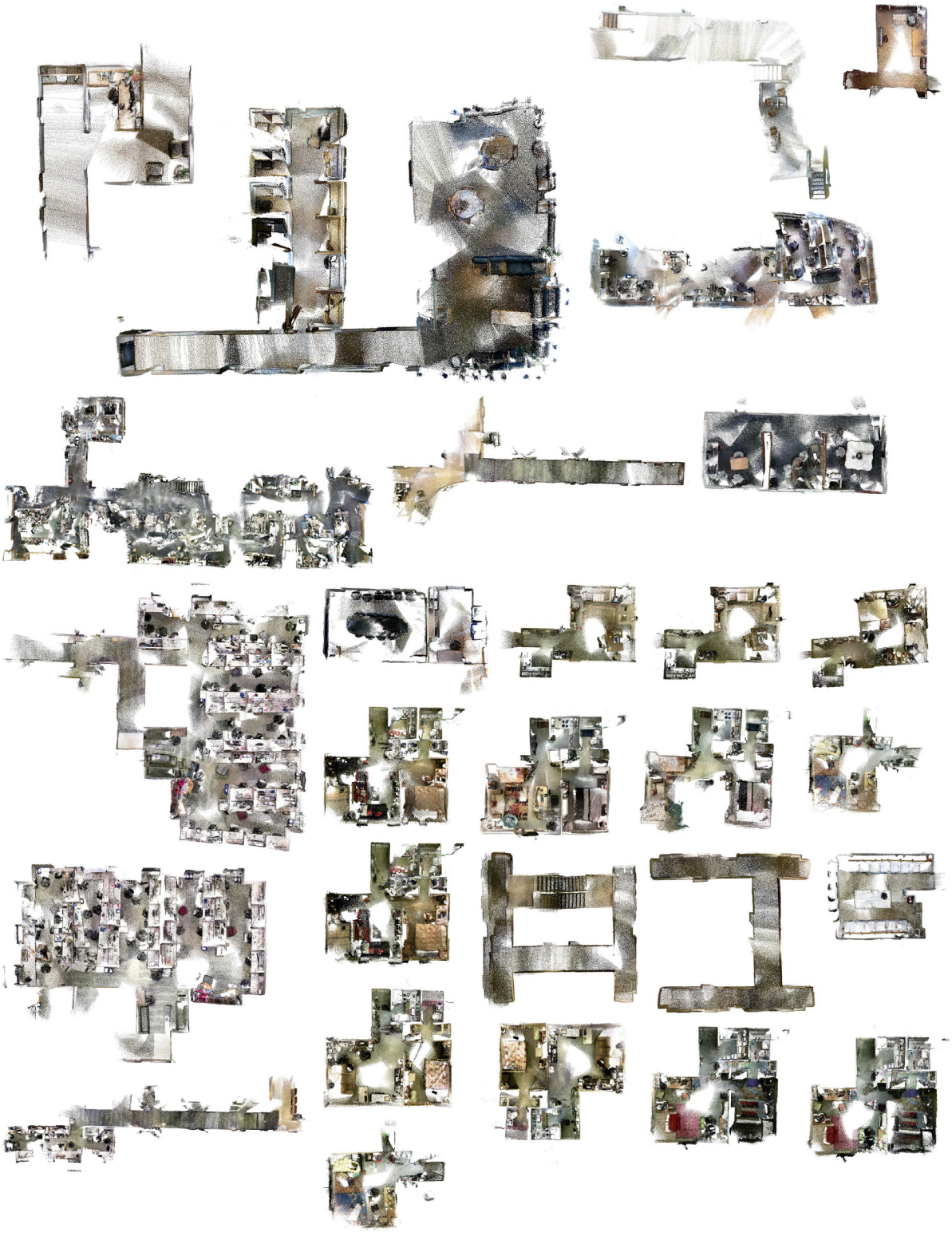

Figure 11: Overhead pointcloud renderings of SUN3D scenes

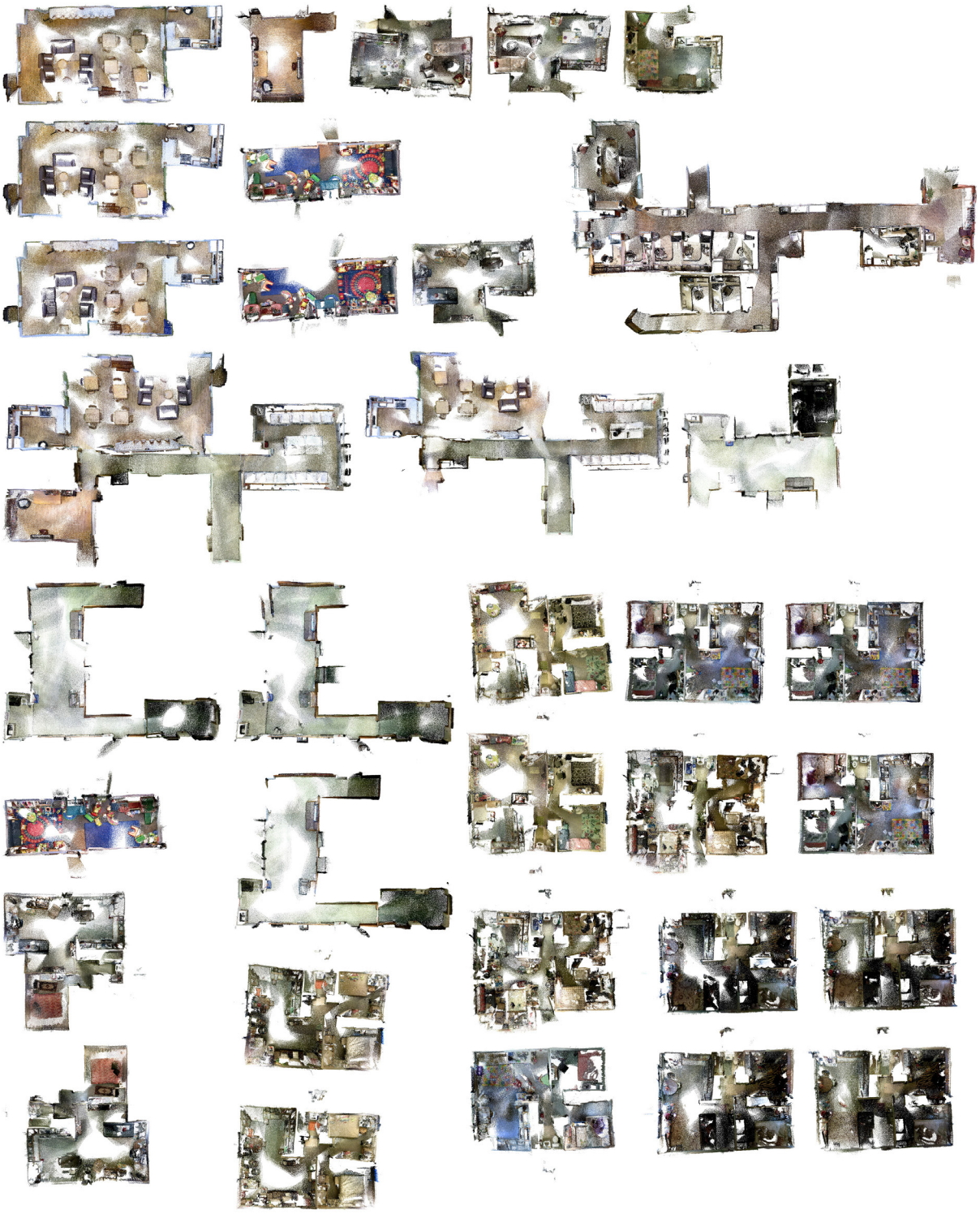

Figure 12: Overhead pointcloud renderings of SUN3D scenes

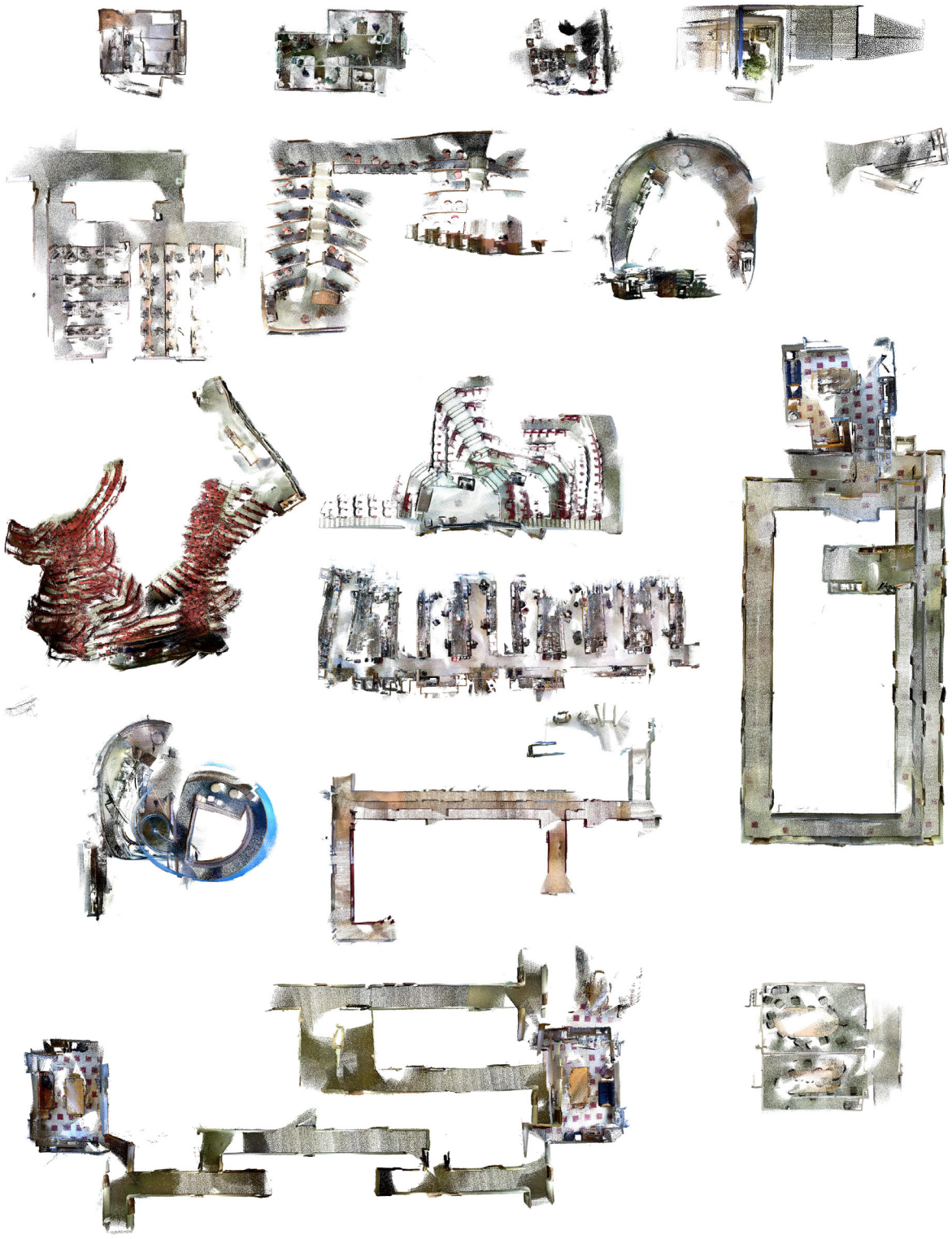

Figure 13: Cases not reconstructing well due to lost tracking in initial alignment. If the transformation between frame i and $i+1$ is highly incorrect, our reconstruction will not be able to recover and will produce erroneous results.

Figure 14: Overhead pointcloud renderings of SceneNN scenes

References

[1] S. Agarwal, K. Mierle, and Others. Ceres solver. http://ceres-solver.org. 2

[2] S. Choi, Q.-Y. Zhou, and V. Koltun. Robust reconstruction of indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3, 4

[3] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *arXiv preprint arXiv:1604.01093*, 2016. 3, 4, 7

[4] T. A. Davis. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006. 2

[5] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung. Scenenn: A scene meshes dataset with annotations. In *International Conference on 3D Vision (3DV)*, 2016. 3

[6] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 2013. 3

[7] K. Pulli. Multiview registration for large data sets. In *Proceedings of the 2Nd International Conference on 3-D Digital Imaging and Modeling*, 3DIM'99, pages 160–168, Washington, DC, USA, 1999. IEEE Computer Society. 3

[8] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012. 3

[9] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. ElasticFusion: Dense SLAM without a pose graph. In *Robotics: Science and Systems (RSS)*, Rome, Italy, July 2015. 3

[10] J. Xiao, A. Owens, and A. Torralba. Sun3d: A database of big spaces reconstructed using sfm and object labels. *Computer Vision, IEEE International Conference on*, 0:1625–1632, 2013. 1, 3