Monte Carlo Scene Search for 3D Scene Understanding Supplementary Material

* Shreyas Hampali[(1)], * Sinisa Stekovic[(1)], Sayan Deb Sarkar[(1)], Chetan S. Kumar[(1)], Friedrich Fraundorfer[(1)], Vincent Lepetit[(2,1)]

[(1)]Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria
[(2)]Université Paris-Est, École des Ponts ParisTech, Paris, France

In this supplementary material:

- we suggest some possible future directions,
- we detail our methods for generating layout and object proposals, and give the pseudocode for MCTS for reference,
- we provide additional comparisons with existing annotations, the results of our MCSS approach, and a baseline using hill climbing for the optimization of our objective function,
- we provide more qualitative results on scans outside the ScanNet dataset.

In addition to this document, we provide a **Supplementary Video** showing the improvement of the solution found by MCSS over time, and additional qualitative demonstrations.

1. Future Directions

While MCSS usually recovers all objects in a scene and complete layouts as we can use low thresholds when generating the proposals without returning false positives, there are still situations where it is challenging to retrieve the correct object models or layout components, when the point cloud misses too much 3D data.

There are still many directions in which our current method could be improved. We could generate proposals from the perspective views as well: RGB images often contain useful information that is missing in the point cloud, and we can handle many proposals. Comparing the final solution with the RGB-D data could also be used to detect objects or layout components that are not explained by the solution, and could be integrated as additional proposals in a new run of MCSS. To improve the 3D poses and models, it would also be interesting to develop a refinement method that improves all the identified objects together.

Furthermore, advanced MCTS-based algorithms such as AlphaZero [7] utilize neural networks to evaluate the quality of state-action pairs. Similarly, it should be possible to train a deep network to predict which proposals should be evaluated first. We thus believe that our approach opens new directions to explore.

*The first two authors contributed equally.

2. Layout Proposal Generation

Figure 1 describes our layout proposal generation. We first detect planes that are likely to correspond to layout components (walls and floors in our experiments). Based on the output from MinkowskiNet [4], we remove from the point cloud the 3D points that do not belong to layout classes, and perform RANSAC plane fitting on the remaining points. We implemented a variant of RANSAC, using 3-point plane fitting that determines inlier-points by their distance and their normals orientation with respect to the sampled plane. We only fit a single floor plane as the SceneCAD dataset [2] does not contain any scenes with multiple floor planes.

At each iteration, our RANSAC procedure fits a plane to three points that are randomly sampled from the remaining point cloud. The inliers are defined as a set of points in the point cloud for which the distance to the plane is less than $10cm$, and the orientation of the normal less than $15°$. We perform 2000 iterations and select the plane with the largest number of inliers. The final inliers are defined by a selection criterion: A set of points in the point cloud for which the distance to the plane is less than $20cm$, and the orientation of the normal is less than $30°$. If the number of inliers of the plane is higher than 5000, we add the plane to the set of layout planes and repeat the RANSAC procedure on the remaining set of outliers. If the number is lower, we perform a second stage RANSAC that seeks to find planes corresponding to small layout components.

In this stage, we set the inlier criterion as follows: A set of points in the point cloud for which the distance to the plane is less than $100cm$, and the orientation of the normal is less than $10°$. The same criterion is used for the final selection. If the number of inliers of the plane is higher than

300, we add the plane to the set of layout planes and repeat the RANSAC procedure on the remaining set of outliers. If the number is lower, we conclude the plane fitting stage.

Then, we proceed to define the set of layout proposals by intersecting the layout planes. More exactly, intersections between non-parallel planes triples are candidate corners for the layout. By connecting the vertices that share a pair of layout planes, we get a set of candidate edges. Finally, by connecting the edges that lie on the same layout plane, we extract a set of valid planar polygons for each of the planes. As the SceneCAD dataset contains only scenes with a single floor level, it is enough to perform the search procedure on wall proposals only: the floor polygon can be directly determined afterwards from the walls. This procedure results in a large number of proposals. For non-cuboid scenes, we obtain between 100 and 1000 proposals, but MCSS can efficiently select the final proposals as shown in Fig. 2.

3. Object Proposal Generation

The synthetic point clouds are generated using the ShapeNet [3] CAD models and the ScanNet [5] dataset. More specifically, we use the instance annotations of ScanNet and replace the point cloud corresponding to each object with a random CAD model from the same category. The complete scenes with the replaced CAD models are rendered into each of the perspective views using the camera poses and are then reprojected back to 3D. This introduces the incompleteness to the synthetic point cloud due to object occlusions. Furthermore, we also introduce depth holes on the rendered depth maps before reprojecting to 3D to make the point clouds more realistic. Fig. 3 shows an example of a synthetic scene.

As explained in Section 4.3 of the main paper and shown in Fig. 4, we use VoteNet [6] and MinkowskiNet [4] to extract the point cloud of each object in the scene. A PointNet++ based network trained on the synthetic point clouds is used for object model retrieval and pose estimation. The model retrieval is performed by regressing the embeddings which are obtained by training a PointNet++ auto-encoder on each category of objects. The pose+scale of the object is obtained by regressing the orientation, bounding box center and size. We use the L2 loss with all the embedding and pose+scale parameters.

In Fig. 5, we show the MCSS tree structure for an example scene constructed from several object proposals.

4. MCSS Pseudocode

MCSS follows the pseudocode for generic MCTS given in Algorithm 1 that is usually used for single-player games. As we explain in the main paper, for the simulation step we can run multiple simulations in practice. For objects, we run 10 simulations in parallel, for layouts we found that running 1 simulation was already enough to achieve robust results.

Algorithm 1: Generic MCTS for non-random single-player games

iters $\leftarrow$ Number of desired runs, best_moves $\leftarrow \varnothing$
while iters > 0 **do**
  $\mathcal{N}_{\text{curr}} \leftarrow \mathcal{N}_{\text{root}}$
  reached_terminal $\leftarrow$ False
  while *not* reached_terminal **do**
    $\mathcal{N}_{\text{curr}} \leftarrow \textsc{Select}(\mathcal{N}_{\text{curr}})$
    if $\mathcal{N}_{curr}$ *is visited for the first time* **then**
      $\textsc{Expand}(\mathcal{N}_{\text{curr}})$
      best_sim $\leftarrow \underset{\text{sim}}{\text{argmax}}\, sc(\textsc{Simulate}(\mathcal{N}_{\text{curr}}, \text{sim}))$
      $\textsc{Update}(\text{best\_sim})$
      if $sc(\text{best\_sim}) > sc(\text{best\_moves})$ **then**
        best_moves $\leftarrow$ moves of best_sim
      reached_terminal $\leftarrow$ True
    else if $\mathcal{N}_{curr}$ *is terminal* **then**
      reached_terminal $\leftarrow$ True
  iters $\leftarrow$ iters - 1
return best_moves

5. Test Scenes used in Scan2CAD Benchmark

There are 2 scenes out of 97 scenes we do not consider from the test set while evaluating on the Scan2CAD benchmark, specifically *scene0791_00* and *scene0793_00*. *scene0791_00* contains multiple floor planes, a special case that we do not address in the object tree, and *scene0793_00* which contains inconsistent manual annotations as the canonical pose of the *chairs* in the ground truth pool are different.

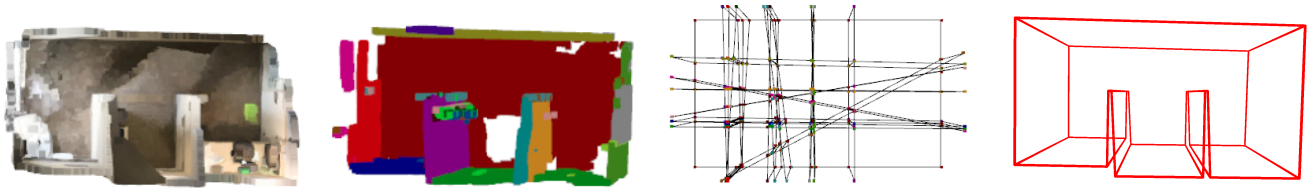
6. Computation Times

For a typical scene with 20 walls and 10 objects, the proposal generation and pre-rendering requires $\sim$15 mins for objects and $\sim$5 mins for layouts. Our MCSS tree search takes 5 mins for 7K iterations on an Intel i7-8700 machine. We would like to point that the proposal generation time especially for objects can be significantly improved by using simplified object models and parallel computations.

7. Comparisons and Visual Results

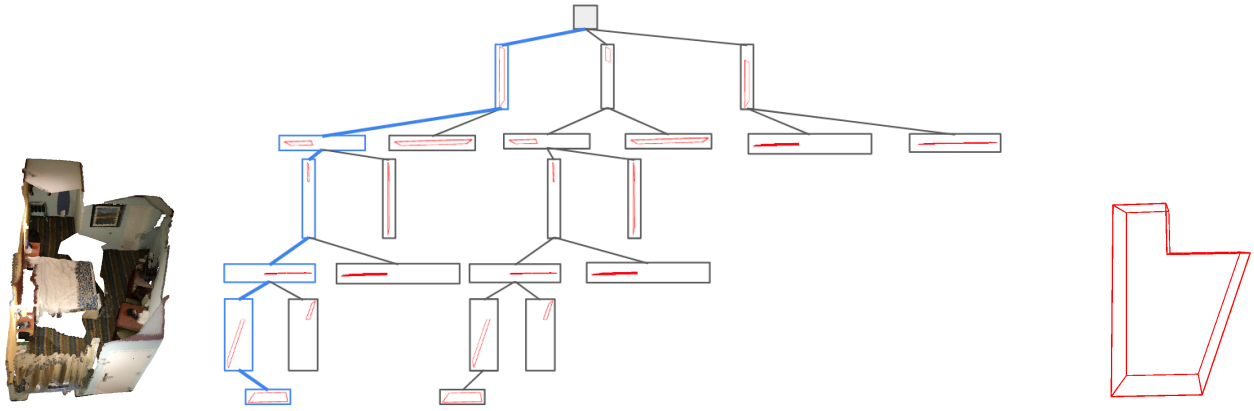
7.1. Hill Climbing Baseline

In addition to the VoteNet baseline for objects (see Section 5.2 of the main paper), for reference, we also compare our method to a more simple hill climbing optimization algorithm than MCSS for both layouts and objects. At each



Input point cloud | Layout planes segmentation | Layout proposals | Reconstructed Layout

Figure 1: We detect layout planes from the input point cloud using our RANSAC procedure. By intersecting these planes, we obtain a large number of planar polygons which we take as our layout proposals. MCSS selects the optimal subset of proposals that best fits the input scene.



Input scene | Layout tree after MCSS | Final solution

Figure 2: The layout proposals are organized into a tree structure such that proposals at the same level of the tree are incompatible to each other but compatible with proposals of their ancestor nodes. Our MCSS approach builds the search tree online and efficiently finds the optimal path, outlined blue, without exploring all candidate solutions.

iteration, the hill climbing algorithm selects the proposal that results in the maximum increase in the scoring function. It stops when no proposal results in an increase. We consider two different scoring functions for the hill climbing algorithm:

- our scoring function $S(\mathcal{O})$ used in MCSS (see Section 4.1 of the main paper). In this case, the selection depends also of the previously selected proposals and the whole images, as the likelihood terms depend on all the image locations. We do not consider proposals that are incompatible with the previously selected proposals.
- the *fitness* of the proposal (see Section 4.2.1 of the main paper). In this case, the scoring function depends mainly on the proposal, but we still use the intersection term in cases of objects, and do not consider proposals that are incompatible with the previously selected proposals.

The hill climbing algorithm is very simple but provides a local minimum.

More generally, most tree search algorithms will prune parts of the tree based on local heuristics. By contrast, MCTS explores the tree up to the leaves, which allows it to look efficiently for the solution based on a global score.

7.2. Layout Estimation

Fig. 6 compares the RGB-D scans, the layout annotations from [2], the layouts retrieved by our MCSS approach, and our new manual annotations for several representative scenes from the ScanNet dataset [5]. We show Scenes *scene0645_00*, *scene0046_00*, *scene0084_00*, *scene0406_00*, and *scene0278_00*. Note that MCSS retrieves detailed layouts, despite noise and missing 3D data.

Fig. 7 shows typical outputs for the hill climbing algorithm. Using our scoring function performs slightly better than simply using the proposals' fitness, however the results are far from perfect as it focuses on the largest components, which may be wrong.

7.3. Objects Retrieval and Pose Estimation

Fig. 8 compares the RGB-D scans, the 3D pose and model annotations from [1], the 3D poses and models re-

Figure 3: An example synthetic point cloud used for training the network which generates the object proposals. The CAD models corresponding to objects are shown on the right.

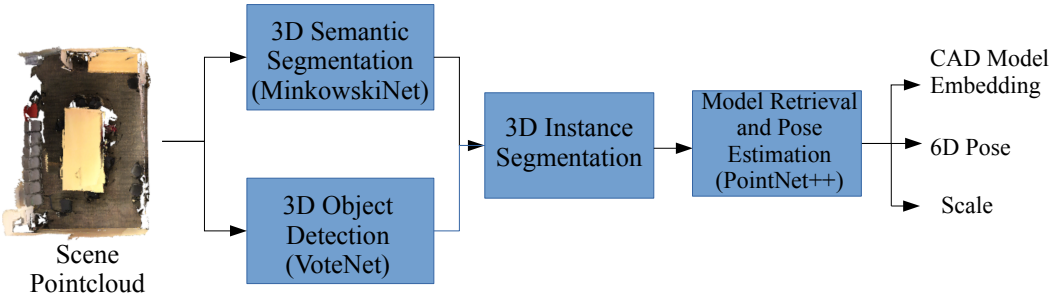


Figure 4: **Object proposals generation pipeline.** We obtain 3D instance segmentation of the input point cloud using the outputs of MinkowskiNet [4] and Votenet [6]. We then retrieve multiple CAD models proposals and their corresponding pose+scale for each object instance using a PointNet++ network, which is trained using synthetic data.

trieved by our MCSS approach, and the output of the VoteNet baseline (see Section 5.2 of the main paper) for several representative scenes from the ScanNet dataset [5]. We show Scenes *scene0249_00*, *scene0549_00*, *scene0690_00*, *scene0645_00*, *scene0342_00*, and *scene0518_00*.

Our method retrieves objects that are not in the manual annotations and sometimes more accurate models: See for example the bed in the 5-th row of Fig. 8. The VoteNet baseline often fails when the objects are close to each other.

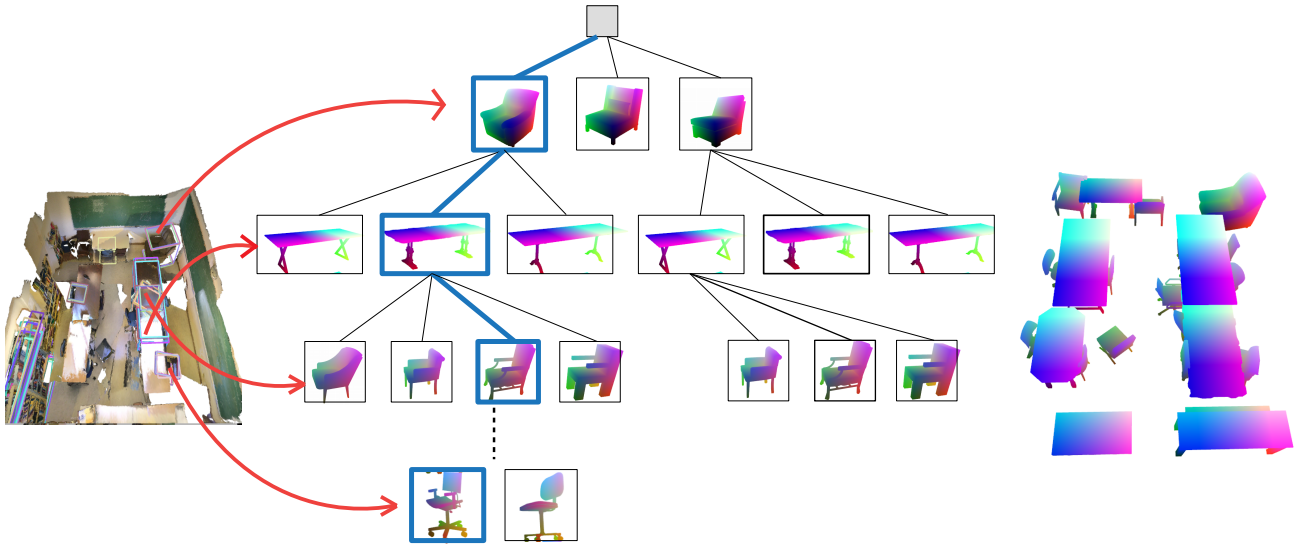
Fig. 9 shows the results of hill climbing, compared to the output of MCSS and manual annotations. The hill climbing algorithm tends to choose large object proposals whenever available, leading to more simplistic solutions that often misses the finer details. Using *fitness* for the scoring function does not consider the occlusions between objects and results in even inferior results.

8. More Qualitative Results

To show that our method can be applied without retraining nor tuning, we scanned additional scene (the authors' office and apartment), and applied MCSS. Fig. 10 shows the scan and the retrieved layouts and objects.

References

[1] Armen Avetisyan, Manuel Dahnert, Angela Dai, Manolis Savva, Angel X. Chang, and Matthias Nießner. Scan2CAD: Learning CAD Model Alignment in RGB-D Scans. In *CVPR*, June 2019. 3, 8



RGB-D Input Object Tree after MCSS Final Solution

Figure 5: **Visualization of an object tree in MCSS.** At each level of the tree, an object proposal is incompatible with other object proposals at the same level, but compatible with the proposal in the parent node and all its ancestors. MCSS builds the search tree online and finds the optimal path, outlined blue, without exploring all the branches of the tree.

[2] Armen Avetisyan, Tatiana Khanova, Christopher Choy, Denver Dash, Angela Dai, and Matthias Nießner. SceneCAD: Predicting Object Alignments and Layouts in RGB-D Scans. In *ECCV*, Aug. 2020. 1, 3, 6

[3] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *CoRR*, abs/1512.03012, 2015. 2

[4] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *CVPR*, 2019. 1, 2, 4

[5] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *CVPR*, 2017. 2, 3, 4, 6, 8

[6] Charles R. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas. Deep Hough Voting for 3D Object Detection in Point Clouds. In *ICCV*, 2019. 2, 4

[7] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play. *Science*, 362(6419):1140–1144, 2018. 1

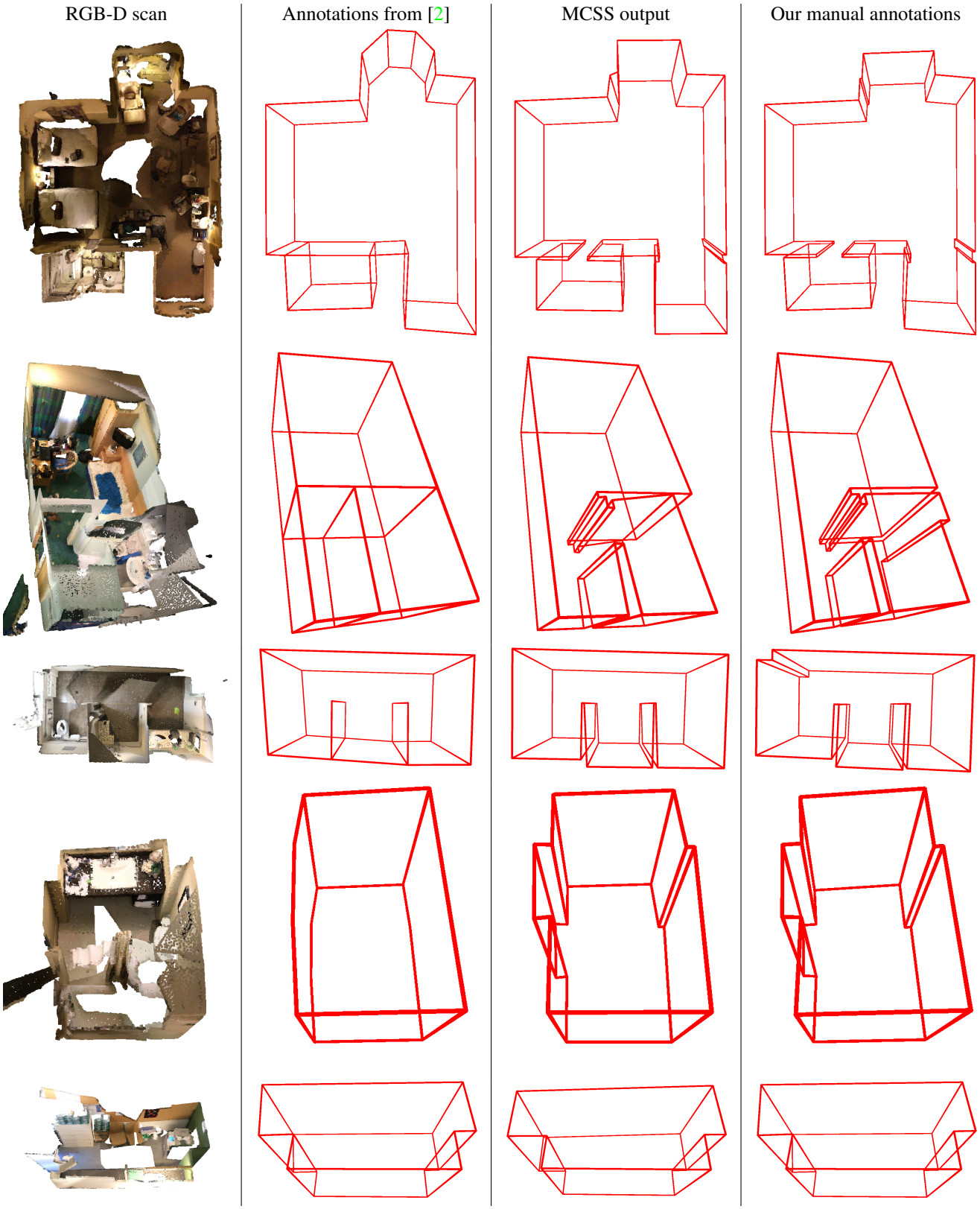


Figure 6: RGB-D scans from ScanNet [5], existing manual annotations, output of our MCSS approach, and our new manual annotations. Note that we retrieve many details despite the noise and missing data in the scans.

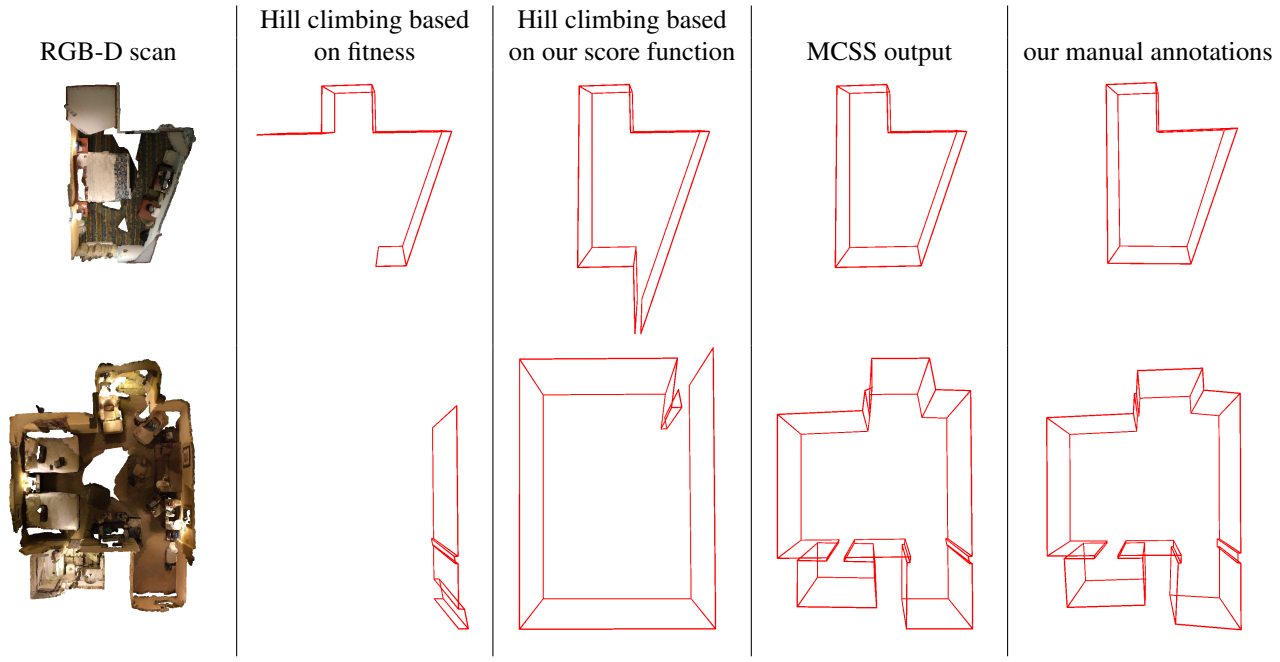


Figure 7: Typical results of the hill climbing optimization for layout estimation and our results. Using our full scoring function slightly helps but the hill climbing algorithm tends to select large components first and cannot recover when they are incorrect. By contrast, our MCSS approach recovers detailed layouts.

Figure 8: RGB-D scans from ScanNet [5], existing manual annotations, output of our MCSS approach, and output of VoteNet for object 3D pose and model retrieval. Note we retrieve objects (shown in red boxes) that are not in the manual annotations, and that VoteNet tends to miss objects or recover an incorrect pose or model when objects are close to each other.

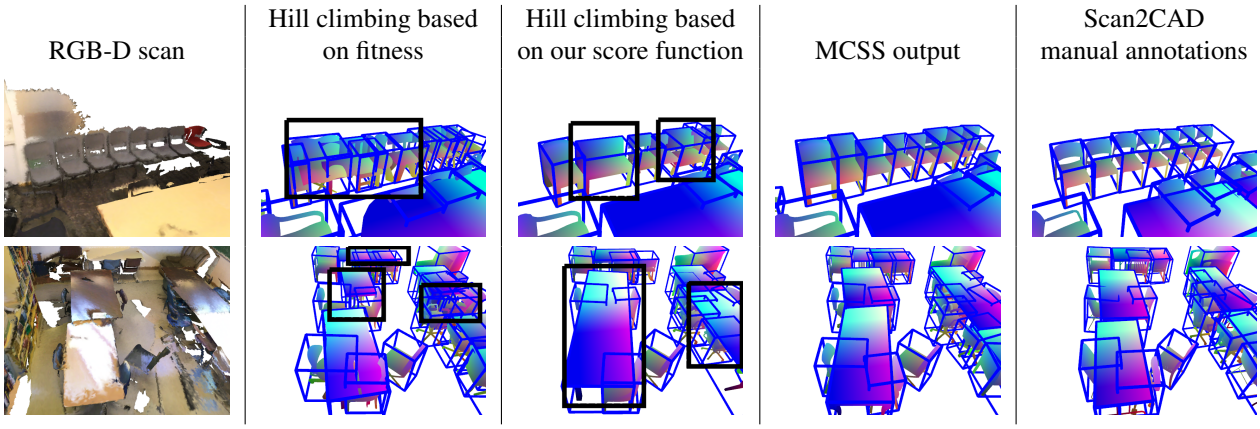
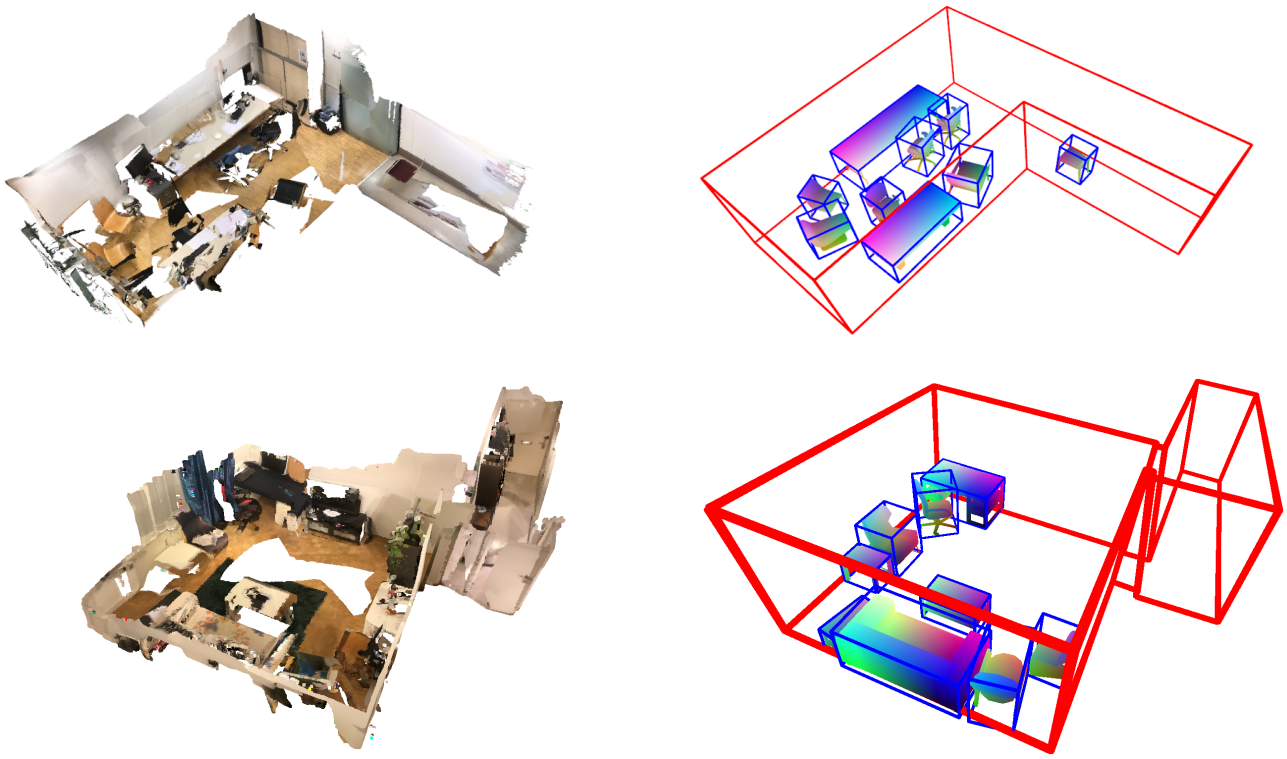


Figure 9: Typical results of the hill climbing optimization for object pose and model retrieval. The Hill climbing algorithm tends to first focus on large object proposals (shown in black boxes), which may be wrong.



(a) (b)

Figure 10: RGB-D scans of the authors' office and apartment (a) and the automatically retrieved object models from the full ShapeNet dataset and layout (b). Our method generalizes well to RGB-D scans outside the ScanNet dataset. Note the large areas with missing data, in particular for the layout.