MonteBoxFinder: Detecting and Filtering Primitives to Fit a Noisy Point Cloud

Michaël Ramamonjisoa[1], Sinisa Stekovic[2], and Vincent Lepetit[1,2]

[1] LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-vallée , France
first.lastname@enpc.fr
[2] Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria sinisa.stekovic@icg.tugraz.at
Project page: https://michaelramamonjisoa.github.io/projects/MonteBoxFinder

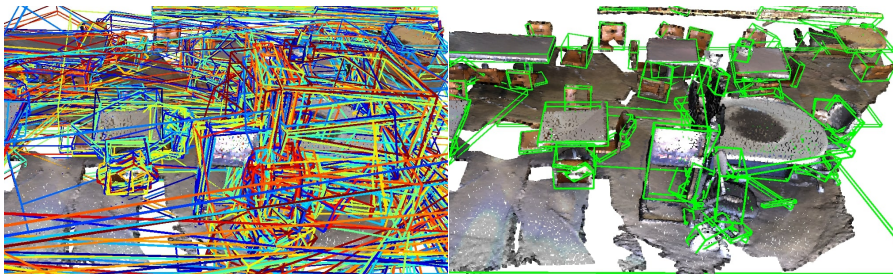

Fig. 1. Given a noisy 3D scan with missing data, our method extracts many possible cuboids, and then efficiently selects the subset that fits the scan best.

Abstract. We present MonteBoxFinder, a method that, given a noisy input point cloud, fits cuboids to the input scene. Our primary contribution is a discrete optimization algorithm that, from a dense set of initially detected cuboids, is able to efficiently filter good boxes from the noisy ones. Inspired by recent applications of MCTS to scene understanding problems, we develop a stochastic algorithm that is, by design, more efficient for our task. Indeed, the quality of a fit for a cuboid arrangement is invariant to the order in which the cuboids are added into the scene. We develop several search baselines for our problem and demonstrate, on the ScanNet dataset, that our approach is more efficient and precise. Finally, we strongly believe that our core algorithm is very general and that it could be extended to many other problems in 3D scene understanding.

Keywords: Primitive Fitting; Discrete Optimization; MCTS

1 Introduction

Representing a 3D scene with a set of simple geometric primitives is a long-standing computer vision problem [24]. Solving it would provide a light representation of 3D scenes that is arguably easier to exploit by many downstream

applications than a 3D point cloud for example. But maybe more importantly, this would also demonstrate the ability to reach a "high-level understanding" of the scene's geometry, by creating a drastically simplified representation.

In this work, we start from a point cloud of a indoor scene, which can be obtained by 3D reconstruction from images or scanning with an RGB-D camera. Recent works have considered representing 3D point clouds with primitives [10,9,22,21]; however they consider "ideal 3D input data", in the sense that the point cloud is complete and noise-free. By contrast, point clouds from 3D reconstruction or scans are typically very noisy with missing data, and robust methods are required to handle this real data.

To be robust to noise and missing data, we propose a discrete optimization-based method. Our approach does not require any training data, which would be very cumbersome to create manually. Given a point cloud, we extract a large number of primitives. While in our experiments we consider only cuboids as our primitives, our approach can be generalized to other choices of primitives. We rely on a simple *ad hoc* algorithm [25] to obtain an initial set of primitives. We expect this algorithm to generate correct primitives but also many false positives. Our problem then becomes the identification of the correct primitives while rejecting the incorrect ones, by searching the subset of primitives that explains the scene point cloud the best.

While the theoretical combinatorics of this search are huge, as they grow exponentially with the number of extracted primitives, the search is structured by some constraints. For example two primitives should not intersect. To tackle this problem, we take inspiration from a recent work on 3D scene understanding [13]. [13] proposes to rely on the Monte Carlo Tree Search (MCTS) algorithm to handle a similar combinatorial problem to select objects' 3D models: The MCTS algorithm is probably best known as the algorithm used by AlphaGo [29]. It is typically used to explore the tree of possible moves in the game Go because it scales particularly well to high combinatorics. [13] adapts it to 3D models selection by considering a move as the selection of a 3D model for one object, and showed it performs significantly better than the simple hill-climbing algorithm that is sometimes used for similar problems [34]. Another advantage of this approach is that it does not impose assumptions on the form of the objective function, unlike other approaches based on graphs, for example [26].

While exploring the solution tree with MCTS as done in [13] is efficient, we show we can still speed up the search for a solution significantly more. The tree structure imposes an ordering of the possible 3D models to pick from. Such sequential structures are necessary when MCTS is applied to games as game moves depend on the previous ones, but we argue that there is a more efficient alternative in the case of object detection and selection for scene understanding.

As illustrated in Figure 2, MCTS works by performing multiple iterations over the tree structure, focusing on the most promising moves. The estimate of how much a move is promising is updated at each iteration. For our problem of primitive selection, we propose to also proceed by iteration. Instead of considering a tree search, at the end of each iteration, we sort the primitives

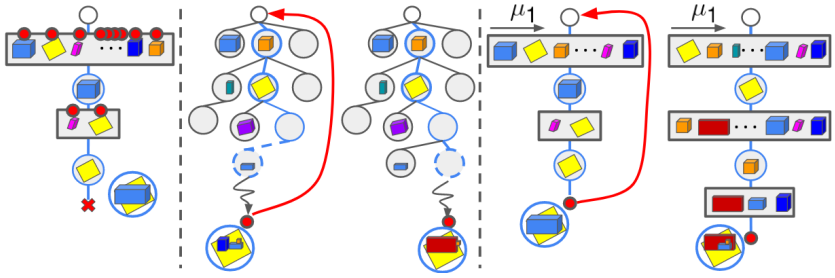

Fig. 2. Comparative overview The hill-climbing algorithm—simply taking the primitive that improves the most the objective function— can terminate × quickly as it gets stuck into a local minimum because of the constraints between primitives. MCTS as used in [13] explores iteratively the solution tree by traversing blue paths, updating which primitives are the most promising ones, but keeping the tree structure fixed. At each iteration, our approach also updates (→) which primitives are the most promising ones, and starts with them. This makes our approach identify a good solution much faster than MCTS in general. Red circles ● represent objective function evaluations. Hill-climbing has to evaluate the complete objective function each time it considers a primitive, while MCTS and our algorithm evaluate the objective function only at the end of an iteration when a complete solution is complete.

according to how likely they are to belong to the correct solution. The next iteration will thus evaluate a solution that integrates the most promising primitives. Our experiments show that this converges much faster to a correct solution.

To evaluate our approach, we experiment on the ScanNet dataset [8], a large and challenging set of indoor 3D RGB-D scans. It contains 3D point clouds of real scenes, with noisy captures and large missing parts, as some parts were not scanned and dark or specular materials are not well captured by the RGB-D cameras. We did not find any previous work working on similar problems, but we adapted other algorithms, namely a simple hill-climbing approach [34] and the MCTS algorithm of [13] to serve as our baselines for comparison. To do so, we introduce several metrics to evaluate the fit quality.

Our algorithm is conceptually simple, and can be written in a few lines of pseudo-code. We believe it is much more general than the cuboid fitting problem. It could first be extended to other type of primitives, and applied to many other selection problems with high combinatorics, and could be applied to other 3D scene understanding problems, for auto-labelling for example. We hope it will inspire other researchers for their own problems.

2 Related Work

In this section, we first discuss related work on cuboid fitting, and then on possible optimisation methods to solve our selection problem.

2.1 Cuboid Fitting on Point Clouds

Primitive fitting is a long standing Computer Vision problem. In the section, we only discuss about methods that operate on point clouds, although there are a large number of methods that are seeking progress in the field of cuboid fitting from 2D RGB images [24,12,18].

Object scale. Sung *et al* [31] leveraged cuboids decompositions to improve 3D object completion of scans of synthetic objects. Tulsiani *et al* [32] introduced object abstraction using cuboids on more challenging objects from the Shapenet [4] dataset. Paschalidou *et al* [22] extended [32] by using the more expressive superquadrics to fit 3D objects. However these methods only operate at the scale of a single objects, on synthetic data, and always assume or are limited to a moderate number of primitives. Some older work related to us have focused on parsing an input point cloud as a decomposition into primitives. Li *et al* [20] decompose a *real* scan of an object into primitives by extracting a set of primitives with RANSAC, which they refine by reasoning on relationship between these primitives. However their method works only on very clean scans, and using object that were *built* as a set of primitives. Furthermore, since they reason about interaction between primitives using a graph, the complexity of there method quickly becomes untractable.

Room-scale cuboid detection. Another class of works has focused on room-scale 3D point cloud parsing with cuboids. A large number of works focused on detecting object bounding boxes in 3D scans have recently emerged since the deep learning era [23,27,28]. Guo *et al* [11] wrote a great survey regarding these methods. Contrary to these methods, our method is able to parse 3D scans with cuboids at the granularity level of parts of objects. Liang *et al* [17] used RGB-D images to fit cuboids to the point cloud obtained by the depth map. In contrast to us, they operate using single-view images, but also leverage color cues via superpixels. Shao *et al* [26] also parse depth maps with cuboids. Given an initial set of cuboids, they build a graph to exploit physical constraints between them to refine the cuboids arrangement. However, they still require human-in-the-loop for challenging scenes, and their graph based method limits the number of cuboids that can be retrieved without exceeding complexity. Our method, in contrast, can deal with number of cuboids that are an order of magnitude larger.

2.2 Solution Search for Scene Understanding

We focus here on scene understanding methods which, like us, do not rely on supervised training data for complete scenes, even if some of them require training data to recognize the objects. These methods typically start from a set of possible hypotheses for the objects present in the scene (similar to the primitives in our case), and choose the correct ones with some optimization algorithms.

Monte Carlo Markov Chain (MCMC) [2] is a popular algorithm to select the correct objects in a scene by imposing constraints on their arrangement.

Method	Uphill	MCTS	Ours
Exploratory	✗	✓	✓
Stochastic	✗	✓	✓
Leverage order invariance	✓	✗	✓

Table 1. Properties of different solution search methods. Our method leverages all popular mechanisms for efficient solution search while leveraging the structure of the problem, which does not require employing tree structures for solution search.

MCMCs can be applied to a parse graph [33,7,15,6] that defines constraints between objects. However, this parse graph needs to be defined manually or learned from manual annotations. Also, MCMCs typically converge very slowly.

Greedy approaches were also used in previous works [16], and they rely on a hill-climbing method to find the objects' poses [16]. [34] selects objects using hill-climbing as well by starting from the objects with the best fits to an RGB-D image. While simple and greedy, this approach can work well on simple scenes. However, it can easily get stuck on complex situations, as our experiments show. [19] uses beam search but this is also an approximation as it also cuts some hypotheses to speed up the search.

Monte Carlo Tree Search (MCTS) was recently used in [13], where they proposed to use MCTS as an optimization algorithm to choose objects that explain an RGB-D sequence. [13] adapts MCTS by considering the selection of one object as a possible move in a game. The moves are selected to optimize an objective function based on the semantic segmentation of the images and the depth maps. The advantage of this approach is that MCTS can scale to complex scenes, while optimizing a complex objective function.

Our approach is motivated by [13]. However, we generate the primitives in a very different way, but more importantly, we propose a novel optimization algorithm, which, contrary to MCTS, does not rely on a tree structure, making it is simpler and significantly more efficient than MCTS, as demonstrated by our experiments.

3 Method

In this section, we first describe how we extract a large pool of cuboids from a given 3D scan. Then, we formalize the selection of the optimal cuboid arrangement. Finally, we detail the solution we propose.

3.1 Generating Cuboid Proposals from Noisy Scans

Figure 3 summarizes our cuboid proposal generation pipeline. The goal of this pipeline is to provide a large pool of cuboids. Some extracted cuboids can be false positives at this stage. The correct subset of cuboids will be selected by the next stage. In this way, we can be robust to noise and missing data in the 3D scan. Our pipeline can be divided in 3 steps: (1) we first extract plane segments;

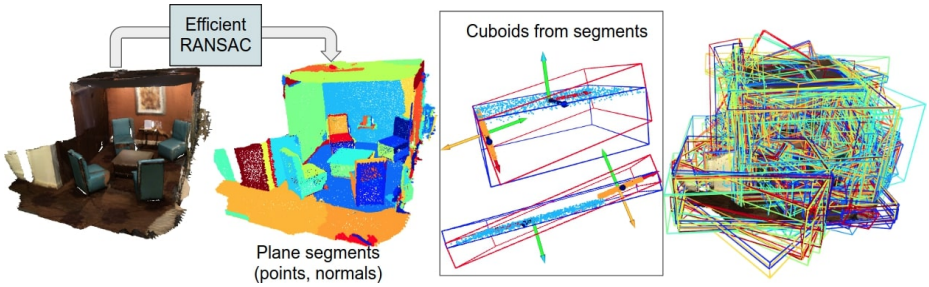

Fig. 3. Overview of our cuboid generation pipeline. After extracting plane segments using an off-the-shelf algorithm [25], we construct cuboids around these segments and pairs of adjacent segments. The result is a dense set of cuboids, which may contain many false positives.

(2) we construct cuboids from pairs of plane segments; (3) we also construct *thin* cuboids by fitting a 3D bounding box to the each plane segment individually. These thin cuboids allow us to represent planar surfaces as well in the final representation. On average, we obtain 880 cuboids and 174 *thin* cuboids per scene.

Extracting planes segments. We use Efficient-RANSAC by Schnabel *et al* [25] to extract 3D planes from the input point cloud. Efficient-RANSAC identifies and returns planar connected components made of 3D points. It is controlled by three hyperparameters: a threshold on the plane-to-point distance to count the inliers, a threshold on the cosine-similarity between normals to points, and a connectivity radius. We use the same hyperparameters for all the scenes in ScanNet, although we could run RANSAC multiple times with various geometric parameters in order to adapt to various types of noise, and still be able to efficiently filter out false positives.

Constructing boxes from pairs of planes. Given a set of planes segments $\{\pi_i = (X_i, \mathbf{N}_i)\}$, where a plane segment π is represented as a point cloud X and its fitted plane normal $\mathbf{N}$, we construct bounding boxes from all pairs of planes (π_A, π_B) that satisfy two criteria, *alignment* and *proximity*. Alignment means that the two normals should be orthogonal or co-linear. Proximity enforces planes segments to have at least one connected component in 3D. We then employ two Gram-Schmidt orthonormalizations to obtain the frame coordinate of two bounding boxes, which are computed to enclose $X_A \cup X_B$. More details can be found in the supplementary material.

Fitting 3D bounding boxes to 3D plane segments. Since we want our method to also retrieve thin objects that may not have compatible neighbors, we therefore fit a 3D oriented bounding box to each plane segment's point cloud X, using the efficient "*Oriented Bounding Box*" method from [5].[3]

[3] We used CGAL's [1] implementation of [25,5].

Algorithm 1: Loss function

procedure `evalObjFunc`(S, Y)

Input : Set of cuboids S, target point cloud Y and its normals $\mathbf{N}(Y)$;

$(X, \mathbf{N}(X)) \leftarrow$ sample_mesh_surface(S);

$\ell_c :=$ ChamferDistance$(X \rightarrow Y)$ + ChamferDistance$(Y \rightarrow X)$;

$\ell_n :=$ CosineDissimilarity$(\mathbf{N}(X) \rightarrow \mathbf{N}(Y))$ + CosineDissimilarity$(\mathbf{N}(Y) \rightarrow \mathbf{N}(X))$;

return $\ell_c \cdot (1 + 0.25 \cdot \exp(\ell_n))$;

3.2 The Cuboids Arrangement Search Problem

We now want to select a subset $\mathcal{S}'$ of $\mathcal{S}$, the set of cuboids generated in Section 3.1, which fits well the input point cloud X of the scene. The cuboids in $\mathcal{S}'$ should not mutually intersect to ensure a minimal representation of this scene.

To solve this problem, we consider (1) an objective function ℓ, defined in Algorithm 1, which will guide the search towards the best solution, and (2) a search algorithm such as the baselines described in Section 3.3, that should be designed to converge to the best solution as efficiently as possible. To better present the algorithms, we introduce a `Cuboid` Class, which we present first.

`Cuboid` Class. We define a `Cuboid` class to instantiate cuboids for our solution search algorithms. It is described by its faces normals and its 8 corners, yielding a surface mesh from which we can sample 3D points. Other attributes can be added to a `Cuboid` , depending on the needs of a particular algorithm, e.g. the number of times a `Cuboid` s has been used in a solution can be denoted as $s.n_1$.

To enforce constraints between cuboids, we need to test if the intersection between two cuboids is small enough. We define this criterion using a variation of the measure of a Intersection-over-Union criterion, and provide its pseudo-code in Supp Mat. `isCompatible`(s_1, s_2, η) measures the ratio between the volume $vol(s_1 \cap s_2)$ of the intersection between both cuboids s_1 and s_2, and the minimum of the volumes of each cuboid $vol(s_1)$ and $vol(s_2)$. In practice, we approximate these volumes by uniformly randomly sampling points from both cuboids and count the points that are inside both s_1 and s_2. The volume ratio is then compared to a threshold η, to decide if the two `Cuboid` intersect. While this test can be performed "*on the fly*" when searching solutions, we pre-compute the pair-wise `Cuboid` compatibility matrix in advance for efficiency.

Objective Function We aim to minimize the distance between our cuboids and the target point cloud, while keeping its normals aligned with the point-cloud's normals. We use Chamfer Distance (CD) and Cosine Dissimilarity, *i.e.* the complement of Cosine Similarity, as our distance and normals deviation losses, yielding full objective function is described in Algorithm 1. In the loss, we truncate CD to $\tau = 0.1$, and normalize it by τ.

3.3 Solution Search Baseline Algorithms

Hill-Climbing Algorithm. The first baseline for our discrete optimization problem is the Hill-Climbing algorithm [30], a naive greedy descent algorithm. This algorithm constructs a solution iteratively, where at each iteration, it comprehensively searches for the proposal that best improves the loss function of a solution $\mathcal{S}_F$, while leaving the solution valid *i.e.* with no incompatibilities. If no proposal is available nor can improve the objective function, the algorithm stops ×. The pseudo code for Hill-Climbing is given in the supplementary material.

MCTS Algorithm. We first describe here the MCTS algorithm, as it inspired our algorithm. [3] provides a full description of the MCTS algorithm. We present it in the context of our cuboid selection problem, following what was done in [13] for 3D model selection. [13] provides a pseudo code for MCTS.

MCTS is able to efficiently explore the large trees that result from the high combinatorics of some games such as Go. As represented in Figure 2, the nodes of the tree correspond to possible states, and the branches to possible moves. MCTS does not build explicitly the entire tree—this would not be tractable anyway—, but only a portion of it, starting from the root at the top.

⇝*Simulation step.* Nodes are thus created progressively at each iteration. To decide which nodes should be created, the existing nodes contain in addition to a state an estimate V of the *value* of this state. To initialize V, MCTS uses a *simulation step* denoted ⇝ in Figure 2, which explores randomly the rest of the tree until reaching a leaf without having to build the tree explicitly. For games, reaching a leaf corresponds to either winning or loosing the game. If the game is won, V should be large; if the game is lost, V should be small.

Adaptation to our problem. Figure 2 shows that in our case, a state in a node is the set of primitives that have been selected so far. A "move" corresponds to adding a primitive to the selected primitives. The children of a node contain primitives that are mutually incompatible, and compatible with the primitives in the ancestor nodes: Such structure ensures that every path in the tree represents a valid solution. In this paper, we consider two possibilities: A varying number of children as in [13] and MCTS-Binary, a binary tree version of MCTS: In MCTS-Binary, a node has two children, corresponding to selecting or skipping a primitive. More details are provided in the supplementary material.

× "Reaching a leaf" happens when no more primitives can be added, because we ran out of primitives or because all the remaining primitives intersect with the primitives already selected. The value V of the new nodes are initialized after the simulation step by evaluating the objective function ● for the set of primitives for the leaf. We take this objective function as a fitness measure between the primitives and the point cloud. Note that this function does not need to have special properties, nor do we need heuristics to guide the tree search.

Selection and expansion steps. At each iteration, MCTS traverses the tree starting from the root node, often using the standard Upper Confidence Bound (UCB) criterion [3] to choose which branch to follow. A high UCB score for a node means that it is more likely to be part of the correct solution. This criterion depends on the values V stored in the nodes and balances exploitation and exploration: When at a node N, we continue with its child node N' that maximizes the UCB score, which depends on the number of times N and N' have been visited so far. This criterion allows MCTS to balance exploration and exploitation.

At some point of this traversal procedure, we will encounter a node with a child node N that has not been created yet, we add the child node to the tree. We use the simulation step described above to initialize $V(N)$ and initialize $n(N)$ to 1.

→*Update step.* MCTS also uses the value $V(N)$ to improve the value estimate of each node N' visited during the tree traversal. Different ways to do so are possible, and we found that for our problem, it is better to take the maximum between the current estimate $V(N')$ and $V(N)$: $V(N') \leftarrow \max(V(N'), V(N))$. $n(N')$, the number of times the node was visited is also incremented.

Final solution. After a chosen number of iterations, MCTS stops. For our problem, we obtain a set of primitives by doing a tree traversal starting from the root node and following the nodes with the highest values V.

3.4 Our algorithm: MonteBoxFinder

We first review the issues when using MCTS for our problem, then give an overview of our algorithm and its components. Finally, we provide some details for each component.

Moving from MCTS. Our primitives selection algorithm is inspired by MCTS, and it is motivated by two observations that show that MCTS is not optimal for our selection problem:

- the order we select the primitives does not matter. However, MCTS keeps growing its tree without modifying the nodes already created. This implies that if a primitive appears at the top of the tree but does not actually belong to the correct solution, it will slow down the convergence of MCTS towards this solution.
- if a node corresponding to adding some primitive P has a high value V, the node corresponding to not keeping P should have a low value, and vice versa. There is no mechanism in MCTS as used in [14] to ensure this. This is unfortunate as one iteration could be used to update more nodes than only the visited nodes.

Algorithm 2: Our MonteBoxFinder Algorithm

Result: Set of selected `Cuboid` $\mathcal{S}_F$
Input: Set of available `Cuboid` $\mathcal{S}$;
Number of evaluations N_{eval} ;
Threshold η ;
Current solution $\mathcal{S}_c := \emptyset$;
Final solution $\mathcal{S}_F := \emptyset$;
Current best loss $\ell^* := +\infty$;
procedure `InitializeNodes(`$\mathcal{S}$`)`
  Input: Pool of `Cuboid` $\mathcal{S}$;
  $\mathcal{S} \leftarrow$ Shuffle$(s \in \mathcal{S})$;
  $\mathcal{S}_c \leftarrow$`Simulate(`$\mathcal{S}$, η`)`;
  $\ell \leftarrow$ `evalObjFunc(`$\mathcal{S}_c$`)`;
  `// Update ALL Cuboid states`
  $\mathcal{S} \leftarrow$ `Update(`$\mathcal{S}$, $\mathcal{S}_c$, ℓ`)`;
  return $\mathcal{S}$

`// MonteBoxFinder Core Algorithm`
$\mathcal{S} \leftarrow$ `InitializeNodes(`$\mathcal{S}$`)`;
for (iter=0; iter$\neq N_{eval}$; iter++) {
  $\mathcal{S} \leftarrow \text{Sorted}_{\downarrow}(s \in \mathcal{S}, s \mapsto s.\mu_1)$;
  $\mathcal{S}_c \leftarrow$`Simulate(`$\mathcal{S}$, η`)`;
  $\ell \leftarrow$ `evalObjFunc(`$\mathcal{S}_c$`)`;
  `// Update ALL Cuboid states`
  $\mathcal{S} \leftarrow$ `Update(`$\mathcal{S}$, $\mathcal{S}_c$, ℓ`)`;
  if $\ell < \ell^*$ **then**
    $\ell^* \leftarrow \ell$;
    $\mathcal{S}_F \leftarrow \mathcal{S}_c$;
return Best solution $\mathcal{S}_F$;

Overview. We give an overview of our algorithm in Algorithm 2. To exploit the two observations described above, we do not use a tree structure. Instead, we use the list of primitives which we sort at each iteration, by exploiting our current estimate for each primitive to be part of the current solution. Our method progressively estimates and exploits a prior probability $\mathcal{P}$ for a primitive to belong to the solution based on our adaptation of the Upper Bounding Criterion (UCB) that balances the exploitation vs. exploration trade-off.

Algorithm 3: Simulate($\rightsquigarrow$) and Update($\rightarrow$) functions of our algorithm

Input: Exploration probability $\mathcal{P}_\epsilon$;
Threshold δ;
procedure `Simulate(`$\mathcal{S}_A$, η`)`
  Input: Pool of available `Cuboid` $\mathcal{S}_A$, threshold η
  Output $\mathcal{S}_F := \emptyset$;
  for ($s \in \mathcal{S}_A$) {
    if s.`isCompatible(`$\mathcal{S}_F$, η`)` **then**
      $\epsilon :=$ uniform_sample$([0, 1])$;
      if $(\epsilon < \mathcal{P}_\epsilon)$ **then**
        if $(s.\mu_1 > s.\mu_0)$ **then**
          $\mathcal{S}_F$.add(s)
      else
        if $(s.\mu_1 < s.\mu_0)$ **then**
          $\mathcal{S}_F$.add(s)
  return $\mathcal{S}_F$

procedure `Update(`$\mathcal{S}$, $\mathcal{S}_F$, ℓ`)`
  Input: Full pool of `Cuboid` $\mathcal{S}$;
  Selected set of `Cuboid` $\mathcal{S}_F \subset \mathcal{S}$;
  Solution score ℓ;
  for ($s \in \mathcal{S}$) {
    if $s \in \mathcal{S}_F$ **then**
      `// Update best` ℓ `when kept`
      $s.\ell_1 \leftarrow \min(\ell, s.\ell_1)$
      $s.n_1 \leftarrow s.n_1 + 1$
      $s.\mu_1 \leftarrow -s.\ell_1 + \sqrt{\ln(1/\delta)/s.n_1}$
    else
      `// Update best` ℓ `when rejected`
      $s.\ell_0 \leftarrow \min(\ell, s.\ell_0)$
      $s.n_0 \leftarrow s.n_0 + 1$
      $s.\mu_0 \leftarrow -s.\ell_0 + \sqrt{\ln(1/\delta)/s.n_0}$
  return $\mathcal{S}$

Initialization. We initialize the run with a few random traversals in order to initialize the states of each `Cuboid` proposal.

Simulate. ($\rightsquigarrow$) At every iteration we first sort primitives $\mathcal{S}_A$ according to their confidence value $s.\mu_1$ in descending order, hence more confident primitives will be more likely selected. Afterwards, we perform the simulation that pops primitives s from sorted $\mathcal{S}_A$. With probability $\mathcal{P}_\epsilon = 0.3$, we perform exploitation and add s to the list of selected proposals $\mathcal{S}_F$ if $(s.\mu_1 > s.\mu_0)$. Otherwise, we perform exploration and add s to $\mathcal{S}_F$ if $(s.\mu_1 < s.\mu_0)$.

UCB Criterion. We modified the UCB score to fit our algorithm, which does not rely on a tree structure. We use this modified term to estimate two confidence measures $s.\mu_0$ and $s.\mu_1$ reflecting how much a cuboid s is likely to belong to the correct solution or not:

$$s.\mu_0 = -s.\ell_0 + \sqrt{\ln(1/\delta)/s.n_0}, \quad s.\mu_1 = -s.\ell_1 + \sqrt{\ln(1/\delta)/s.n_1}, \tag{1}$$

where $s.\rho_0$ and $s.\rho_1$ are the minimum loss values reached when rejecting and accepting primitive s, $s.n_0$ and $s.n_1$ denote the number of times that the primitive were rejected and selected respectively, and $\delta = 0.03$ is a hyperparameter modifying the exploration rate, smaller δ implies larger exploration.

Update ($\rightarrow$) In comparison with the *update step* of MCTS described in 3.3, our MonteBoxFinder algorithm updates *all* primitives states after an iteration. If a primitive s was selected, we update its $s.\ell_1$, $s.\mu_1$, and $s.n_1$ values based on the obtained loss ℓ and our adapted UCB criterion, otherwise we update its $s.\ell_0$, $s.n_0$, and $s.\mu_0$ values instead. In the next iteration during simulation, we use these value to determine whether to select or reject the primitive.

4 Experiments

4.1 Dataset

ScanNet [8] is a dataset that contains noisy 3D scans of 1613 indoor scenes. We evaluate our method on the full dataset, where for each scene, we used the decimated and cleaned point clouds provided in [8] both for the box proposals generation step and for the solution search step.

4.2 Metrics

Fitness measures. The most direct way to measure the quality of a solution is to measure the loss function ℓ described in Algorithm 1. Indeed, we want to evaluate the ability of our algorithm to search the solution space. Additionally, we measure a bi-directional precision metric Pr_τ. $\mathrm{Pr}_\tau$is computed as the proportion of points successfully matched between the "*synthetic*" point cloud X,

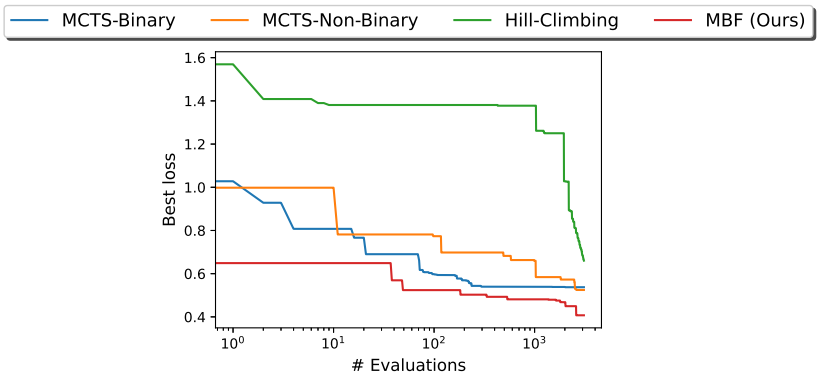

Fig. 4. Value of the objective function for the best found solution as a function of the number of evaluations for Hill-Climbing, MCTS, MCTS-Binary, and our MonteBoxFinder (MBF) method. Hill-Climbing requires many evaluations before finding a reasonable solution, which explains the flat curve at the beginning. It also gets stuck into a local minimum and stops improving. In this experiment, we give the number of evaluations Hill-Climbing used before getting stuck to the three other methods. Our method converges significantly faster than the other methods towards a better solution. Similar graphs for other scenes are provided in the supplementary.

generated by sampling 3D points from retrieved 3D cuboid meshes, and the 3D scan Y. A point is successfully matched if its Chamfer Distance (CD)[4] value is below a threshold $\tau = 0.2$:

$$\mathrm{Pr}_\tau = \frac{|\{x \in X \text{ s.t. } CD[x \to Y] \leq \tau\}|}{2|X|} + \frac{|\{y \in Y \text{ s.t. } CD[y \to X] \leq \tau\}|}{2|Y|} . \quad (2)$$

Efficiency measure. The motivation for developing our approach compared to [13] is to converge faster towards a good solution. In order to measure efficiency of a given method, we consider the curve of the objective function of the best found solution as a function of the iteration, as the ones showed in Figure 4. We use the Area Under the Curve (AUC) given a maximum budget of iterations N_{eval}: the lower the AUC, the faster the convergence. We also report AUC (norm), which normalizes the AUC values of the different between 0 and 1, with 0 being the value of the best performing method and 1 being the value of the worst performing method.

Complexity measure. We observe that bad solutions tend to contain a small number of selected primitives. This is because it is challenging to find a large subset of cuboids with no intersection between any pair of cuboids. Hence we also report the number of cuboids in the retrieved solutions.

[4] In this case, we do not apply the normalization discussed in Algorithm 1

	Loss↓	Precision ↑	AUC ↓	AUC (norm) ↓	Avg. # Cuboids ↑
Hill-Climbing	0.383	0.928	0.871	0.998	12
MCTS	0.247	0.966	0.427	0.225	28
MCTS-Binary	0.292	0.961	0.370	0.102	35
Ours (MonteBoxFinder)	**0.201**	**0.982**	**0.322**	**0.018**	**37**

Table 2. Comparison between our method and our baselines. Our method outperforms all baselines on all metrics computed on ScanNet. We retrieve a more accurate fit, while being able to find more non-intersecting cuboids.

4.3 Evaluation Protocol

For all scenes from the ScanNet dataset [8], we run the Hill-Climbing method, and obtain its solution $\mathcal{S}_{\text{HC}}$. We then consider the number N_{eval} of evaluations of the objective that were required by Hill-Climbing to construct this solution. We then run MCTS and our algorithm using the same number of evaluations N_{eval}. This ensures the three methods are compared fairly, as they are given the same evaluation budget, which is by far the most costly step of all three algorithms.

4.4 Quantitative results

Table 2 provides the results of our experimental comparisons. As expected, the Hill-Climbing algorithm performs worst: By greedily selecting proposals that minimize the loss, it gets stuck to local minimum solutions consisting of large proposals. It can also provide a complete solution only once it converged, while MCTS, MCTS-Binary and our method can provide a good solution much faster. The table also shows that our algorithm converges significantly faster than MCTS and MCTS-Binary, which was the desired goal. Interestingly, MCTS-Binary performs better than the original MCTS method of [13]. In the supplementary material, we discuss in details the links between our method and MCTS-Binary.

4.5 Qualitative Results

Figure 5 shows qualitative results. Hill-climbing focuses on large cuboids to describe the scene. MCTS often selects many true positives but misses some of the proposals because it cannot explore deeper levels of the tree for the given iteration budget. In contrast, our algorithm is able to successfully retrieve cuboid primitives for objects of different sizes, such as walls, floors, and furniture.

5 Conclusion

We proposed a method for efficiently and robustly finding a set of cuboids that fits well a 3D point cloud, even under noise and missing data. Our algorithm is

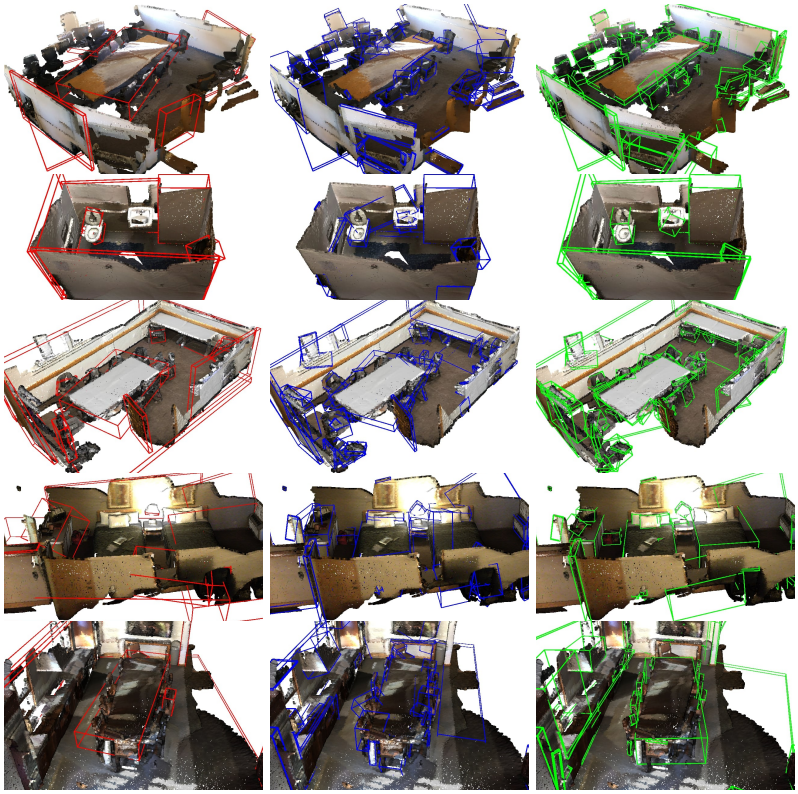

Fig. 5. Qualitative results. Hill-climbing often selects large cuboids that span across multiple different objects (first, third, fourth rows, and fifth rows). MCTS does better, but does not sufficiently explore the solution space (second row). In contrast, our algorithm outperforms both methods and is able to successfully reconstruct many chairs in first, third, and fifth rows, and bedroom furniture in fourth row. *More qualitative results are provided in the supplementary material.*

not restricted to cuboids, and could consider other primitives. Only a procedure to identify the primitives is required, even if it generates many false positives as our algorithm can reject them. Moreover, the output of our algorithm could be used to generate labeled data for training a deep architecture for fast inference. This could be done to predict cuboids from point clouds, but also from RGB-D images, since the 3D scans of ScanNet were created from RGB-D images By simply reprojecting the cuboids retrieved by our method, we can obtain RGB-D images annotated with the visible cuboids.

Acknowledgments We would like to thank Pierre-Alain Langlois for his suggestions and help with CGAL. We thank Gul Varol, Van Nguyen Nguyen and Georgy Ponimatkin for our helpful discussions. This project has received funding from the CHISTERA IPALM project.

References

1. CGAL User and Reference Manual
2. Andrieu, C., Freitas, N.D., Doucet, A., Jordan, M.I.: An Introduction to MCMC for Machine Learning. Machine Learning (2003)
3. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez liebana, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games **4:1**, 1–43 (2012)
4. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An Information-Rich 3D Model Repository. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015)
5. Chang, C.T., Gorissen, B., Melchior, S.: Fast Oriented Bounding Box Optimization on the Rotation Group $SO(3, R)$. ACM Trans. Graph. **30**(5) (oct 2011). https://doi.org/10.1145/2019627.2019641, `https://doi.org/10.1145/2019627.2019641`
6. Chen, Y., Huang, S., Yuan, T., Qi, S., Zhu, Y., Zhu, S.C.: Holistic++ Scene Understanding: Single-View 3D Holistic Scene Parsing and Human Pose Estimation with Human-Object Interaction and Physical Commonsense. In: International Conference on Computer Vision (ICCV) (2019)
7. Choi, W., Chao, Y.W., Pantofaru, C., Savarese, S.: Understanding Indoor Scenes Using 3D Geometric Phrases. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2013)
8. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
9. Deprelle, T., Groueix, T., Fisher, M., Kim, V., Russell, B., Aubry, M.: Learning Elementary Structures for 3D Shape Generation and Matching. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 7433–7443 (2019)
10. Groueix, T., Fisher, M., Kim, V.G., Russell, B., Aubry, M.: AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
11. Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L., Bennamoun, M.: Deep learning for 3d point clouds: A survey. IEEE transactions on pattern analysis and machine intelligence **43**(12), 4338–4364 (2020)
12. Gupta, A., Efros, A.A., Hebert, M.: Blocks World Revisited: Image Understanding Using Qualitative Geometry and Mechanics. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) European Conference on Computer Vision (ECCV). pp. 482–496. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
13. Hampali, S., Stekovic, S., Sarkar, S.D., Kumar, C.S., Fraundorfer, F., Lepetit, V.: Monte Carlo Scene Search for 3D Scene Understanding. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
14. Hampali, S., Stekovic, S., Sarkar, S.D., Kumar, C.S., Fraundorfer, F., Lepetit, V.: Monte Carlo Scene Search for 3D Scene Understanding. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
15. Huang, S., Qi, S., Zhu, Y., Xiao, Y., Xu, Y., Zhu, S.C.: Holistic 3D Scene Parsing and Reconstruction from a Single RGB Image. In: European Conference on Computer Vision (ECCV) (2018)

16. Izadinia, H., Shan, Q., Seitz, S.M.: Im2CAD. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
17. Jiang, H., Xiao, J.: A Linear Approach to Matching Cuboids in RGBD Images. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2013)
18. Kluger, F., Ackermann, H., Brachmann, E., Yang, M.Y., Rosenhahn, B.: Cuboids Revisited: Learning Robust 3D Shape Fitting to Single RGB Images. In: International Conference on Computer Vision and Pattern Recognition (CVPR). pp. 13070–13079 (June 2021)
19. Lee, D.C., Gupta, A., Hebert, M., Kanade, T.: Estimating Spatial Layout of Rooms Using Volumetric Reasoning About Objects and Surfaces. In: Advances in Neural Information Processing Systems (NeurIPS) (2010)
20. Li, Y., Wu, X., Chrysanthou, Y., Sharf, A., Cohen-Or, D., Mitra, N.J.: GlobFit: Consistently Fitting Primitives by Discovering Global Relations. ACM Transactions on Graphics **30**(4), 52:1–52:12 (2011)
21. Paschalidou, D., van Gool, L., Geiger, A.: Learning Unsupervised Hierarchical Part Decomposition of 3D Objects from a Single RGB Image. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
22. Paschalidou, D., Ulusoy, A.O., Geiger, A.: Superquadrics Revisited: Learning 3D Shape Parsing beyond Cuboids. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
23. Qi, C.R., Litany, O., He, K., Guibas, L.J.: Deep hough voting for 3d object detection in point clouds. In: Proceedings of the IEEE International Conference on Computer Vision (2019)
24. Roberts, L.G.: Machine Perception of Three-Dimensional Solids. Ph.D. thesis, Massachusetts Institute of Technology (1963)
25. Schnabel, R., Wahl, R., Klein, R.: Efficient RANSAC for Point-Cloud Shape Detection. Comput. Graph. Forum **26**, 214–226 (06 2007). https://doi.org/10.1111/j.1467-8659.2007.01016.x
26. Shao, T., Monszpart, A., Zheng, Y., Koo, B., Xu, W., Zhou, K., Mitra, N.: Imagining the Unseen: Stability-based Cuboid Arrangements for Scene Understanding. ACM SIGGRAPH Asia 2014 (2014), * Joint first authors
27. Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., Li, H.: Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
28. Shi, W., Rajkumar, R.: Point-gnn: Graph neural network for 3d object detection in a point cloud. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1711–1719 (2020)
29. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature **529**(7587), 484–489 (2016)
30. Skiena, S.: The Algorithm Design Manual. Springer (2010)
31. Sung, M., Kim, V.G., Angst, R., Guibas, L.: Data-driven Structural Priors for Shape Completion. ACM Transactions on Graphics (Proc. of SIGGRAPH Asia) (2015)
32. Tulsiani, S., Su, H., Guibas, L.J., Efros, A.A., Malik, J.: Learning Shape Abstractions by Assembling Volumetric Primitives. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2017)

33. Zhao, Y., Zhu, S.C.: Scene Parsing by Integrating Function, Geometry and Appearance Models. In: International Conference on Computer Vision and Pattern Recognition (CVPR) (2013)
34. Zou, C., Guo, R., Li, Z., Hoiem, D.: Complete 3D Scene Parsing from an RGBD Image. IJCV (2019)