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

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

1 Supplementary Video

In addition to this supplementary document, we provide a supplementary video showing progress through iterations for our method and compared baselines.

2 More on Cuboids

2.1 Constructing Cuboids from Pairs of Plane Segments

In this section, we provide more details regarding the construction of cuboids given a pair of plane segments $\pi_A = (X_A, \mathbf{N}_A)$ and $\pi_B = (X_B, \mathbf{N}_B)$.

Checking planes adjacency We recall that two planes (π_A, π_B) should be considered for constructing a cuboid only if they fulfill two requirements: *alignment* and *proximity*.

Proximity Two plane segments are adjacent if they verify the *proximity* criterion, which requires that they have at least one connected component, such that $\min(\text{ChamferDistance}(X_A \to X_B), \text{ChamferDistance}(X_B \to X_A)) < \gamma$, where γ is a small 3D distance.

Alignment Two plane segments $\pi_A = (X_A, \mathbf{N}_A)$ and $\pi_B = (X_B, \mathbf{N}_B)$ are aligned if they are either "orthogonal enough" or "co-linear enough"; This corresponds to $|\mathbf{N}_A^T \mathbf{N}_B| < \alpha$ or $|\mathbf{N}_A^T \mathbf{N}_B| > \beta$, respectively, where $\alpha << 1$ and $\beta \lesssim 1$.
In our experiments we set $(\alpha, \beta, \gamma) = (0.3, 0.7, 0.05m)$.

Getting cuboids main axes We can now construct *two* orthonormal bases $\mathcal{B}_A = (\mathbf{u}_A, \mathbf{v}_A, \mathbf{w}_A)$ and $\mathcal{B}_B = (\mathbf{u}_B, \mathbf{v}_B, \mathbf{w}_B)$ of vectors using Gram-Schmidt orthonormalization with N_A or N_B as the first vector alternatively, as shown in Equations (1a) and (1b).

$$\mathbf{u}_A := \frac{\mathbf{N}_A}{\|\mathbf{N}_A\|_2}$$
$$\mathbf{v}_A := \mathbf{N}_B - \frac{\mathbf{u}_A^T \mathbf{N}_B}{\|\mathbf{N}_B\|_2}\mathbf{u}_A \quad (1a)$$
$$\mathbf{v}_A \leftarrow \frac{\mathbf{v}_A}{\|\mathbf{v}_A\|_2}$$
$$\mathbf{w}_A := \mathbf{u}_A \times \mathbf{v}_A,$$

$$\mathbf{u}_B := \frac{\mathbf{N}_B}{\|\mathbf{N}_B\|_2}$$
$$\mathbf{v}_B := \mathbf{N}_A - \frac{\mathbf{u}_B^T \mathbf{N}_A}{\|\mathbf{N}_A\|_2}\mathbf{u}_B \quad (1b)$$
$$\mathbf{v}_B \leftarrow \frac{\mathbf{v}_B}{\|\mathbf{v}_B\|_2}$$
$$\mathbf{w}_B := \mathbf{u}_B \times \mathbf{v}_B,$$

Computing the final cuboids Based on the two cuboids bases, we compute their sizes by simply projecting all 3D points $X \in X_A \cup X_B$ and computing the minimum and maximum of the projections along each axes of $\mathcal{B}_A$ and $\mathcal{B}_B$. This results in two bounding boxes aligned with $\mathcal{B}_A$ and $\mathcal{B}_B$ respectively, which both enclose all points in $X_A \cup X_B$.

2.2 Computing Intersections with `isCompatible`

In order to check compatibility between cuboids s_1 and s_2, we design a variation of an Intersection-over-Union criterion, replacing the *Union* with the volume of the smallest cuboid between s_1 and s_2. In order to compute the volume of the intersection, we approximate volumes by sampling points in both s_1 and s_2 and counting points that are inside both. Full details of the procedure used in `isCompatible` are given in Algorithm 1. In practice we use an intersection threshold $\eta = 10\%$.

3 More About our Baselines

3.1 The Hill-Climbing Algorithm

The Hill-Climbing algorithm [1] is a naive greedy descent algorithm that constructs a solution iteratively, where at each iteration, it comprehensively searches for the proposal that best improves the objective 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 ×. A pseudo-code for the Hill-Climbing algorithm is provided in Algorithm 2.

Sub-optimal solutions In practice, Hill-Climbing leads to sub-optimal solutions where the algorithm gets stuck into a local minimum. This is because the algorithm first greedily fits large regions of the scene, therefore employing large `Cuboid` ; This makes a lot of potentially good cuboids unavailable, as they would intersect with that large `Cuboid` .

Algorithm 1: The `isCompatible` function

procedure `isCompatible`(s_1, $\mathcal{S}$, η)
  Result: Returns True if `Cuboid` s_1 is compatible with all `Cuboid` in $\mathcal{S}$
  Input : Cuboids `Cuboid` $\mathcal{S}$, threshold η
  if ($\forall s_2 \in \mathcal{S}$, `IntersectionOverVolume`($s_1$,$s_2$)$> \eta$) **then**
    return False ;
  return True ;
procedure `IntersectionOverVolume`(s_1, s_2)
  Input : Cuboids s_1 and s_2
  Volume of `Cuboid` s_1 V_1 := Volume(s_1)
  Volume of `Cuboid` s_1 V_2 := Volume(s_2)
  Number of samples $N_{samples}$:= 5000;
  Number of samples from s_1 in s_2 $N_{1\subset 2}$:= 0;
  Number of samples from s_2 in s_1 $N_{2\subset 1}$:= 0;
  // Sample 3D points within both cuboids s_1 and s_2
  $\mathcal{X}_1$:= sample_points_inside(s_1, N) ;
  $\mathcal{X}_2$:= sample_points_inside(s_2, N) ;
  // Count points sampled in s_1 which are also inside s_2
  for ($x \in \mathcal{X}_1$) {
    if $x \in s_2$ **then**
      $N_{1\subset 2} \leftarrow N_{1\subset 2} + 1$;
  // Count points sampled in s_2 which are also inside s_1
  for ($x \in \mathcal{X}_2$) {
    if $x \in s_1$ **then**
      $N_{2\subset 1} \leftarrow N_{2\subset 1} + 1$;
  // Compute approximation of the intersection volume
  Intersection := $\frac{V_1 \cdot N_{2\subset 1} + V_2 \cdot N_{1\subset 2}}{2N_{samples}}$;
  return $\frac{\text{Intersection}}{\min(V_1, V_2)}$;

Sub-optimality of evaluations Hill-climbing has to evaluate the complete objective function ● *each time* it considers a primitive, which is particularly costly at the beginning of the algorithm where the solution $\mathcal{S}_F$ is still empty, since no `Cuboid` candidate would intersect with it. After selecting a large `Cuboid` , the set of available, *i.e.* compatible cuboids gets dramatically reduced, hence resulting in an acceleration of the search of Hill-Climbing as shown in Figure 4 of main paper, which however converges to sub-optimal solutions. In contrast, MCTS and our algorithm evaluate the objective function only at the end of an iteration when a complete solution is complete.

3.2 Binary-Tree MCTS

In our non-binary tree adaptation of MCTS, the search algorithm spends many iterations on iterating the first levels of the tree which might contain many, mutually incompatible, primitives. Therefore, this can limit the exploitation ca-

Algorithm 2: Hill-climbing algorithm

Result: Set of selected `Cuboid` $\mathcal{S}_F$
Input: Set of proposal `Cuboid` $\mathcal{S}$;
Threshold η ;
Final solution $\mathcal{S}_F := \emptyset$;
Current best loss $\ell^* := +\infty$;
Current best `Cuboid` $s^* := \emptyset$;
Set of available `Cuboid` $\mathcal{S}_A := \mathcal{S}$;
Evaluations counter $N_{eval} := 0$;

```
while S_A ≠ ∅ do
    s* ← ∅;
    for ( s ∈ S_A ) {
        if s.isCompatible(s_f, η) then
            S_F.add(s);
            ℓ ← evalObjFunc(S_F);
            N_eval ← N_eval + 1;
            if ℓ < ℓ* then
                ℓ* ← ℓ;
                s* ← s;
            S_F.remove(s);
        else
            S_A.remove(s);
    S_F.add(s*);
return S_F, N_eval
```

pabilities of MCTS as the algorithm prioritizes nodes that have not been visited yet. In our binary-tree adaptation of MCTS, every level of the tree corresponds to selecting or skipping a primitive. The resulting tree structure, hence, trades tree-breadth for tree-depth, which enables better exploitation. However, due to a large depth of the tree, MCTS does not explore solutions in the bottom of the tree, hence we observed only minor improvements over its non-binary adaptation. We argue that our MonteBoxFinder method can be seen as an adaptive version of binary-tree MCTS. In contrast to binary-tree MCTS, as we show in Figure 1, the tree equivalent of our method is able to adapt its structure during the search and enable better update mechanism leading to faster convergence.

4 More results

4.1 Qualitative results

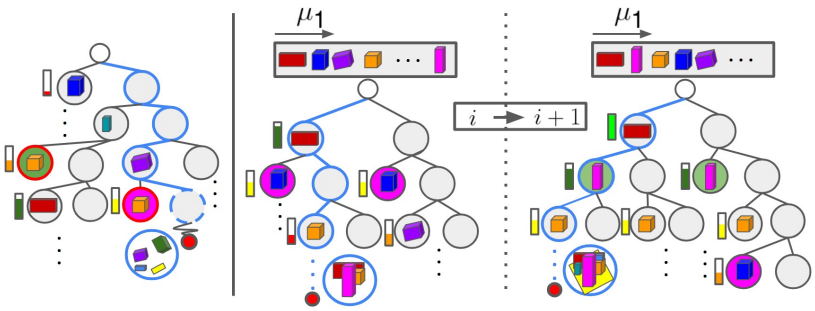
We show more qualitative results in Figure 2.

4.2 Progress plots

We show more progress plots in Figure 3.

References

1. Skiena, S.: The Algorithm Design Manual. Springer (2010)



Binary-tree MCTS 2 iterations of our algorithm interpreted with a binary tree

Fig. 1. We observe that behavior of our algorithm can be interpreted as an adaptive binary-tree MCTS, even though we do not explicitly define a tree structure. As MCTS is bound by its tree structure, it will invest iterations into exploring primitives in the upper part of the tree, even those with low confidence, visualized as **colored bars**. Further more, as indicated with **colored circles**, MCTS models confidences of same primitives in different parts of the tree independently. Blue circles indicate a selection path of a single MCTS iteration that fails to extract meaningful proposals due to aforementioned difficulties. In contrast, our method sorts at each iteration primitives according to their confidences μ_1 and will focus more easily on more promising primitives. In addition, as indicated by **colored circles** we only model a single confidence per primitive. These features enable our method to converge faster to good solutions in practice.

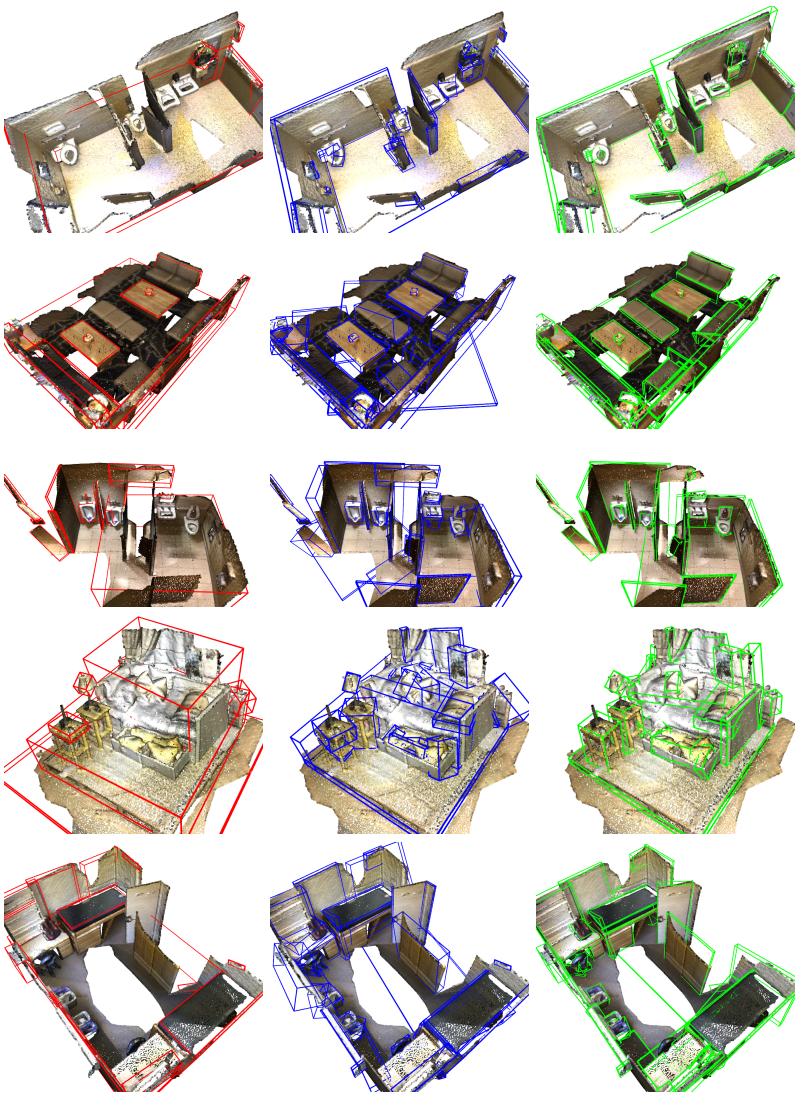


Fig. 2. Qualitative results. Hill-climbing often selects large cuboids that span across multiple different objects. MCTS does better, but sometimes yields outliers (second and fifth row). In contrast, our algorithm outperforms both methods and is able to successfully reconstruct many more details.

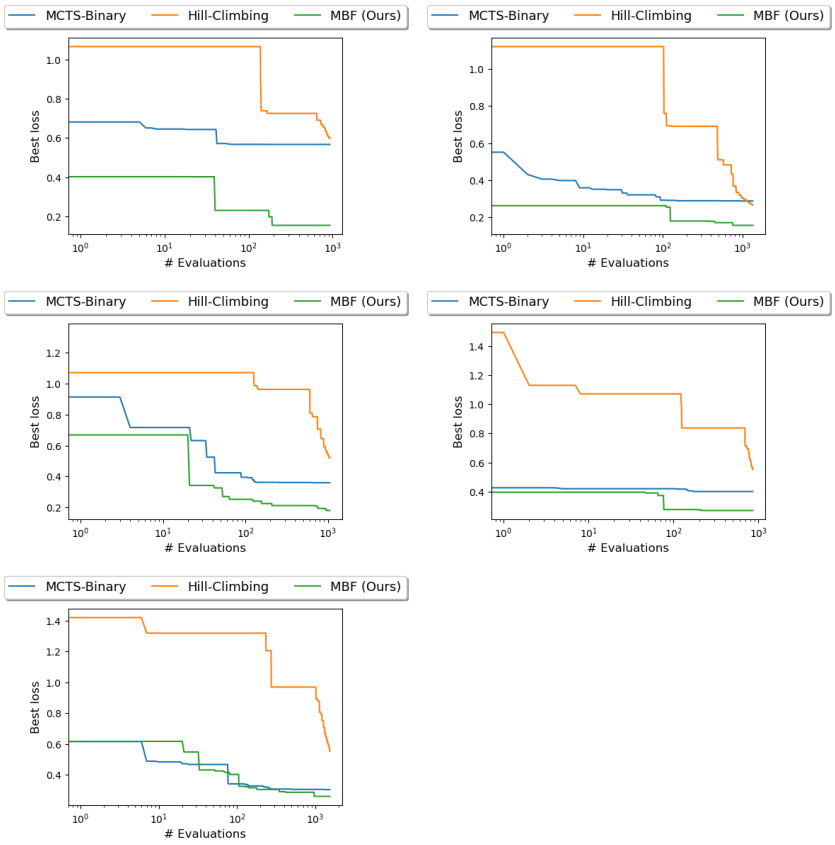


Fig. 3. Samples of progress plots. Our method consistently outperforms its baselines, i.e. the Hill-Climbing algorithm and MCTS, as it converges faster to a better solution. These plots correspond to the scene examples in Figure 2