PyTorchGeoNodes: Enabling Differentiable Shape Programs for 3D Shape Reconstruction – Supplementary Material

Sinisa Stekovic[1], Stefan Ainetter[1], Mattia D'Urso[1], Friedrich Fraundorfer[1], and Vincent Lepetit[2]

[1] Inst. for Computer Graphics and Vision, Graz Univ. of Technology, Graz, Austria
[2] LIGM, École des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France
{sinisa.stekovic, stefan.ainetter, fraundorfer}@icg.tugraz.at
mattia.durso@tugraz.at vincent.lepetit@enpc.fr

We provide additional evaluations of our search algorithm in Section 1, and provide more details regarding our PyTorchGeoNodes framework in Section 2, regarding our search algorithm in Section 3, and regarding our shape program designs in Section 4.

In addition, we provide a video showing the reconstruction progress of our approach through MCTS iterations for scenes of the ScanNet dataset [2].

1 Additional Validation

In this section, we provide additional results to show that our method is applicable to different shape programs. Tables 1, 2, 3 show quantitative evaluations for the 'Sofa', 'Chair', and 'Table' categories. Table 4 shows evaluations for the 'Table' and 'Cabinet' categories for scenes from the ScanNet dataset. Note that in the case of 'Cabinet', we discarded 'Has Back' and 'Number of Dividing Boards' from the ScanNet evaluations as they could not be quantitatively evaluated: This is because the large majority of cabinets have drawers, therefore the back of cabinets are not visible and we cannot estimate the number of dividing boards based on geometric objective terms in the presence of drawers. In the case of bookshelves, objects on shelves induce noise in instance segmentation. Therefore, the objective term does not handle such settings well. We note these are general limitations for reconstructing cabinets and are present also for state-of-the-art methods [1]. Qualitative results in Figure 1, 2 show additional examples where our approach is able to reconstruct a variety of 3D shapes that are geometrically consistent with the input scene.

	Parameter	Full Method	No Refinement	No Refinement No Exploitation Term
		Mean Absolute Difference to Ground Truth (↓)		
Continuous Parameters	Width [cm]	**8.042**	16.301	15.761
	Height [cm]	**2.304**	4.012	4.646
	Depth [cm]	**6.214**	6.792	7.128
	Leg Size [cm]	**1.623**	1.711	1.711
	Arm Width [cm]	**5.383**	6.478	6.478
	Arm Height [cm]	**8.235**	8.84	9.807
	Arm Depth [cm]	**12.769**	14.747	14.863
	Back Height [cm]	**2.488**	5.127	5.531
	Back Depth [cm]	**5.1**	6.813	6.813
	L Width [cm]	**4.317**	5.341	5.341
	L Depth [cm]	**11.495**	12.879	12.502
	Rotation [°]	**3.252**	26.559	26.63
		Classification Accuracy (↑)		
Discrete Parameters	Has Legs	68.0	**70.0**	60.0
	Has Left Arm	**88.0**	65.0	64.0
	Has Right Arm	**89.0**	65.0	60.0
	Has Arm Legs	**62.7**	56.0	52.0
	Has Back	**99.0**	97.0	98.0
	Is L-Shaped	**81.0**	**81.0**	**81.0**
	Flip L Around Y	**93.6**	53.2	63.8

Table 1: Accuracy of the retrieved shape parameters for 'Sofa' on synthetic scenes. For continuous parameters, we report the mean absolute difference to the ground truth parameters. For discrete parameters, we report the classification accuracy. We provide the results for the three variants described in Section 5.3 of the main paper.

	Parameter	Full Method	No Refinement	No Refinement No Exploitation Term
		Mean Absolute Difference to Ground Truth (↓)		
Continuous Parameters	Legs Size [cm]	**1.239**	1.484	1.484
	Leg Offset [cm]	3.483	**2.667**	6.562
	Bottom Thickness [cm]	**1.278**	1.602	1.602
	Bottom Size Scale	**3.35**	4.926	6.767
	Seat Height [cm]	**1.971**	3.332	2.853
	Seat Width [cm]	**1.793**	4.089	4.559
	Seat Depth [cm]	**3.802**	5.049	5.049
	Seat Thickness [cm]	**1.08**	1.333	1.333
	Back Height [cm]	**5.525**	7.329	10.273
	Backrest Scale	**0.12**	**0.12**	0.22
	Back Thickness [cm]	**1.296**	1.424	1.424
	Backrest Offset Scale	**0.19**	0.25	0.29
	Arm Depth Scale	**0.78**	0.83	0.92
	Arm Height [cm]	**3.401**	4.579	4.579
	Arm Width [cm]	**1.708**	1.845	1.845
	Arm Thickness [cm]	**0.766**	0.777	0.777
	Rotation [°]	**0.506**	17.148	17.192
		Classification Accuracy (↑)		
Discrete P.	Legs Type	**100.0**	**100.0**	**100.0**
	Has Leg Support	**96.3**	94.4	90.7
	Has Back	95.0	**97.0**	96.0
	Has Arms	**94.0**	90.0	81.0

Table 2: Accuracy of the retrieved shape parameters for 'Chair' on synthetic scenes. For continuous parameters, we report the mean absolute difference to the ground truth parameters. For discrete parameters, we report the classification accuracy. We provide the results for the three variants described in Section 5.3 of the main paper.

	Parameter	Full Method	No Refinement	No Refinement No Exploitation Term
Continuous Parameters		Mean Absolute Difference to Ground Truth (↓)		
	Width	**17.842**	33.853	35.956
	Height	**2.361**	3.671	2.67
	Depth	**16.923**	24.328	26.387
	Top Thickness	**1.132**	1.471	1.471
	Mid Leg X Scale	**0.17**	0.2	0.21
	Mid Leg Y Scale	**0.15**	0.19	0.21
	Mid Board Z Scale	**0.16**	**0.16**	**0.16**
	Rotation	**1.462**	24.316	27.207
Discrete P.		Classification Accuracy (↑)		
	Top Shape	**90.5**	64.0	60.0
	Legs Type	**87.8**	68.0	70.0
	Has Mid Board	**100.0**	96.2	96.2

Table 3: Accuracy of the retrieved shape parameters for 'Table' on synthetic scenes. For continuous parameters, we report the mean absolute difference to the ground truth parameters. For discrete parameters, we report the classification accuracy. We provide the results for the three variants described in Section 5.3 of the main paper.

		Classification accuracy [%] (↑)			Chamfer
		Top Shape	Legs Type	Has Mid Board	Dist. [cm] (↓)
Table	Random SP	50.65	32.04	47.87	58.6
	w/o refinement	93.78	50.67	82.03	11.8
	Full (Ours)	**96.52**	**52.17**	**84.92**	**11.7**
		Has Legs	Has Drawers		
Cabinet	Random SP	54.167	43.75		19.7
	w/o refinement	56.25	**68.75**		**0.6**
	Full (Ours)	**62.5**	64.583		0.9

Table 4: Results of shape reconstruction for 'Cabinet' and 'Table' categories on the ScanNet scenes. An interesting observation is that in contrast to 'Sofa', 'Chair', and 'Table', we did not observe significant advantages when adding the refinement step to our search algorithm for the 'Cabinet' category. Even without refinement, MCTS performs very well which is indicated by a very low chamfer distance. The accuracy for 'Has Drawers' is somewhat low considering that it influences the 3D shape considerably. The shape of cabinets is ambiguous in terms of drawers, especially when we estimate the rotation as well. As the back side of cabinets is never shown in scenes from ScanNet, rotating cabinets by 180° will still result in geometrically consistent reconstruction.

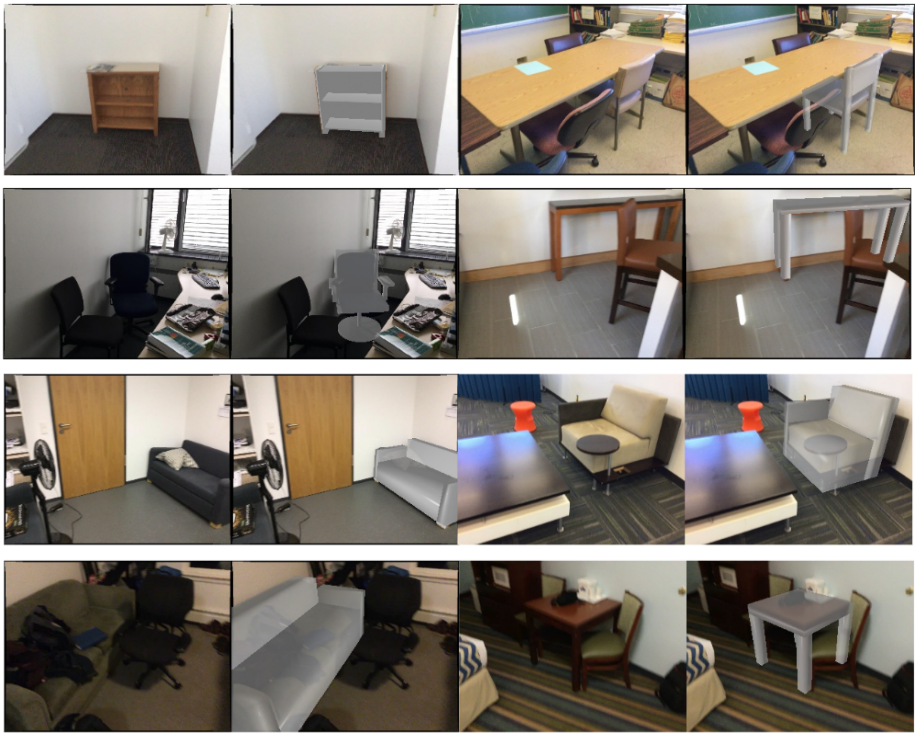

Fig. 1: Additional qualitative results show that our method generalizes to a variety of scenarios. We observe that for all examples, retrieved models are geometrically consistent with the scene.

Fig. 2: Additional qualitative results show that our method generalizes to a variety of scenarios. The last row shows two failure cases, first a minor failure due to limitations in the design of our shape program for the chair category, and second, our reconstructed cabinet model is rotated by 180°. We observe that for all examples, retrieved models are geometrically consistent with the scene.

2 PyTorchGeoNodes – Implementation Details

In this section, we provide additional details regarding the implementation of our PyTorchGeoNodes framework.

2.1 Computational Graph

Shape programs in PyTorchGeoNodes are represented as computational graphs. Therefore, for every functionality in the main paper, PyTorchGeoNodes implements a class with the corresponding functionality.

These functionalities are implemented in the form of nodes and edges:

- Nodes in our graphs are child classes of PyTorch base class `torch.nn.Module` to enable seamless integration into PyTorch code. Every node in the graph is associated with a unique id. When performing a forward pass, a node can take either default constants or outputs of other nodes as input. Therefore, nodes can contain multiple input and output sockets to enable flow of information through the computational graph.
- Edge is a data structure with four attributes. 'Input node' and 'Output node' define the identifiers of individual nodes that are connected by the edge. 'Input socket' and 'Output socket' define the corresponding sockets. Therefore, a node can be associated with several input and output edges, and an edge is always shared between exactly two nodes.

During a forward pass, we use a hash map that keeps track of the output sockets of individual nodes such that they can be easily accessed by nodes by simply querying the correct hash. Every 'Input node' in the graph parses named parameters, or shape parameters, and initializes the hash map that is updated with each forward call of nodes in the graph. Finally, we accumulate outputs of 'Output nodes' as a list of output geometries. Note that in our experiments we only consider computational graphs that have one output.

2.2 Efficient Implementation

As computational graphs increase in size (in our experiments, graphs can have between 100 and 200 nodes) so does the computational time which is why we considered different ways to improve the efficiency of PyTorchGeoNodes.

Note that, by default, computational nodes are not necessarily pre-sorted in optimal order. A node could request an input that has not been computed yet. As this would lead to several complications and the usage of recursive calls, which would in turn lead to computational bottlenecks, we implement a different solution. During the generation of the computational graph, nodes are sorted into a list using topological sort: Based on dependencies in a graph, we ensure that a node always appears after nodes that it is dependent on. For example, 'Input nodes' and other nodes that do not require any inputs will appear first, and 'Output nodes' will be the last nodes after sorting. Then during inference, we

can simply iterate this list, as inputs for nodes will be readily available once the forward pass of the nodes is invoked.

In addition, we observed that some nodes in a graph do not necessarily depend on any inputs. For example, instantiating an initial geometric primitive does not necessarily depend on any parameters. In this case, we implement caching such that the output of such nodes does not need to be recomputed during every forward pass.

3 Search Algorithm – Implementation Details

In this section, we discuss implementation details regarding our search algorithm.

Discretizing continuous parameters. When organizing our search tree, along with discrete shape parameters we also include discretized values for continuous parameters. For a given valid range of a parameter, we generate discrete values in linear steps of 0.1. This value is fitting for all parameters since they are represented in meters, or as aspect ratio relative to other parameters. In the case of object rotation, we use discrete steps of size 45°.

Shape parameter tree. As briefly mentioned in the main paper, it is beneficial to organize the search tree in such a way that geometrically more similar objects are part of the same subtree. The order in which different shape parameters appear in the tree is defined based on the influence of the parameter on the final geometry. More exactly, first, we randomly initialize shape parameters $\mathcal{P}$ and generate the corresponding initial 3D shape $\mathrm{Sh}(\mathcal{P})$. Then for a parameter p and its discrete values $v_p \in p$, we modify the initial shape parameters $\mathcal{P}(v_p)$, generate the 3D shape $Sh(\mathcal{P}(v_p))$, and calculate the chamfer distance CD to the initial 3D shape. Then, the influence of a shape parameter is calculated as:

$$\mathcal{I}(P) = \sum_{v_p \in p} \big(\mathrm{CD}(Sh(\mathcal{P}), Sh(\mathcal{P}(v_p))) - \mu_{CD}\big)^2, \tag{1}$$

where μ_{CD} is the mean of chamfer distances for parameter p. We average $\mathcal{I}$ over 100 instances of initial shapes to obtain the final influence of the parameter. This equation ensures that both geometric influence, as well as the number of individual discrete values for a parameter, are taken into account when determining the order of nodes in a tree.

Search details. As discussed in the main paper, MCTS depends on several different hyper-parameters. Based on empirical observations from our synthetic experiments, we use the following configuration:

- Total number of iterations is set to 300;
- Number of simulations is set to 50. Setting this number high helps avoid local minima in early iterations of MCTS. However, this number is only used in early iterations of MCTS, as every time a simulation selects a solution that was already visited, we conclude the simulation as it is likely that this subtree is already well-explored;
- We set $\lambda = 0.2$ to weight the exploration term during the selection phase;

- We use the Adam optimizer during the optimization phase with the learning rate set to $5e^{-2}$;
- Optimization is the computational bottleneck in our search algorithm. Therefore, we only perform optimization if a leaf node is reached during the selection phase, and skip optimization if the solution is reached during the simulation phase of MCTS . This way, we make sure that we only optimize good solutions.

4 Shape Program Designs for Validation

Here, we provide more details on our designs of individual shape programs.

'Cabinet' consists of 12 parameters, 8 continuous, 3 boolean, and 1 integer parameter:

- 'Width', 'Height', and 'Depth' are continuous parameters that control the width , height and depth of a cabinet in meters. The valid ranges of values are $[0.3, 2.0]$, $[0.3, 2.5]$ and $[0.1, 0.6]$, respectively;
- 'Board Thickness' is a continuous parameter that controls the thickness of side boards. The valid range of values is $[0.01, 0.09]$;
- 'Has Back' is a boolean parameter that controls whether a cabinet has a back board;
- 'Has Legs' is a boolean parameter that controls whether a cabinet has legs;
- 'Leg Width', 'Leg Height', 'Leg Depth' are continuous parameters that control the width, height, and depth of legs in meters. The valid range is $[0.03, 0.1]$.
- 'Number of Dividing Boards' is an integer parameter that controls the number of dividing boards on a cabinet. The valid range of values is in $[2, 5]$;
- 'Dividing Board Thickness' is a continuous parameter that controls the thickness of dividing boards in meters. The valid range of values is in $[0.01, 0.05]$;
- 'Has Drawers' is a boolean parameter that controls whether the cabinet has drawers.

'Chair' consists of 20 parameters, 16 continuous, and 4 boolean parameters:

- 'Legs Type' is a boolean parameter that determines whether a chair has four legs or one leg in the middle.
- 'Legs Size' is a continuous parameter that determines the thickness of a chair in meters. The valid range of values is in $[0.02, 0.08]$;
- 'Has Leg Support' is a boolean parameter that controls whether legs of a four-legged chair are connected with support elements;
- 'Support Offset' is a continuous parameter that controls the height offset of leg support relative to the seat height. The valid range of values is in $[0, 0.5]$;
- 'Bottom Thickness' is a continuous parameter that controls the thickness of the bottom cylindric surface of one-legged chairs in meters. The valid range of values is in $[0.02, 0.08]$;
- 'Bottom Size Scale' is a continuous parameter that controls the radius of the bottom surface relative to seat width. The valid range is in $[0.7, 1.0]$;

- 'Seat Height', 'Seat Width', 'Seat Depth', 'Seat Thickness' are continuous parameters that control the geometry of the seat in meters. The valid ranges are $[0.3, 0.9]$, $[0.4, 0.8]$, $[0.4, 0.6]$ and $[0.04, 0.1]$, respectively;
- 'Has Back' is a boolean parameter that controls whether a chair has back elements;
- 'Back Height' is a continuous parameter that controls the height of the chair back in meters. The valid range is in $[0.3, 1.0]$;
- 'Backrest Scale' is a continuous parameter that scales length of backrest relative to the height of the backrest. The valid range is in $[0.1, 1.0]$;
- 'Back Thickness' is a continuous parameter that controls the thickness of the back elements in meters. The valid range is in $[0.02, 0.08]$;
- 'Backrest Offset Scale' is a continuous parameter that positions the backrest relative to the height of the back. The valid range is in $[0.0, 1.0]$;
- 'Has Arms' is a boolean parameter that controls whether a chair has arms;
- 'Arm Depth Scale' is a continuous parameter that controls the arm depth relative to the seat depth. The valid range is in $[0.5, 0.8]$;
- 'Arm Height' is a continuous parameter that controls the height of arms in meters. The valid range is in $[0.1, 0.3]$;
- 'Arm Width' is a continuous parameter that controls the width of arms in meters. The valid range is in $[0.08, 0.15]$;
- 'Arm Thickness' is a continuous parameter that controls the thickness of arms in meters. The valid range is in $[0.02, 0.05]$.

'Sofa' consists of 18 parameters, 11 continuous parameters and 7 boolean parameters:

- 'Width', 'Height', 'Depth' are continuous parameters that control width, height and depth of the base of a sofa in meters. The valid ranges are in $[0.5, 2.7]$, $[0.3, 0.6]$ and $[0.3, 0.6]$, respectively;
- 'Has Legs' is a boolean parameter that controls whether the sofa has legs;
- 'Leg Size' is a continuous parameter that controls the size of legs in meters. The valid range is in $[0.03, 0.1]$;
- 'Has Left Arm' and 'Has Right Arm' are boolean parameters that control whether the sofa has arms;
- 'Arm Width', 'Arm Height', 'Arm Depth' are continuous parameters that control the width, height, and depth of arms in meters. The valid ranges are in $[0.05, 0.3]$, $[0.5, 0.8]$, $[0.6, 1.0]$;
- 'Has Arm Legs' is a boolean parameter that controls whether a sofa has legs directly under the arms;
- 'Has Back' is a boolean parameter that controls whether a sofa has a back;
- 'Back Height' and 'Back Depth' are continuous parameters that control the height and depth of the back in meters. The valid ranges are in $[0.3, 0.7]$ and $[0.05, 0.3]$;
- 'Is L-Shaped' is a boolean parameter that controls whether the sofa contains the 'L' extension;
- 'L Width' and 'L Depth' control the width and depth of the 'L' extension in meters. The valid ranges are in $[0.3, 0.5]$, $[0.3, 1.0]$;

– 'Flip L Around Y' is a boolean parameter that controls whether the 'L' extension is on the left or the right side of the couch.

'Table' consists of 10 parameters, 6 continuous, 2 boolean and 1 integer parameters:

– 'Width', 'Depth' and 'Height' are continuous parameters that control the width, depth, and height of a table in meters. The valid ranges are in $[0.4, 4.0], [0.4, 1.3], [0.4, 1.5]$;
– 'Top' is a boolean that controls whether the shape of the top board of a table is cylindrical or cuboidal;
– 'Top thickness' is a continuous parameter that controls the thickness of the top in meters. The valid range is in $[0.04, 0.1]$;
– 'Legs Type' is an integer parameter that controls whether a table has 1 in the middle, or 2 legs, or 4 legs on the side.
– 'Mid Leg X Scale' and 'Mid Leg Y Scale' are continuous parameters that scale the middle leg relative to the width and depth of the table. The valid range for both parameters is in $[0.05, 1.0]$;
– 'Has Mid Board' is a boolean parameter that controls whether a table has a second board underneath the top;
– 'Mid Board Z Scale' is a continuous parameter that controls the offset of the middle board relative to the height of the table. The valid range is in $[0.05, 0.5]$.

References

1. Ainetter, S., Stekovic, S., Fraundorfer, F., Lepetit, V.: HOC-Search: Efficient CAD Model and Pose Retrieval from RGB-D Scans. International Conference on 3D Vision (2024)
2. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In: Conference on Computer Vision and Pattern Recognition (2017)