

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

Sinisa Stekovic^{1,2} Arslan Artykov¹ Stefan Ainetter² Mattia D’Urso² Friedrich Fraundorfer²

¹ LIGM, École des Ponts et Chaussees, IP Paris, CNRS, France

² Inst. for Computer Graphics and Vision, Graz Univ. of Technology, Austria

Project page: vevenom.github.io/pytorchgeonodes

We provide additional results and details regarding evaluations of our search algorithm and other baselines in Section 1, and provide more details regarding our PyTorchGeoNodes framework in Section 2, regarding our search algorithm in Section 3, regarding our integration of Gaussian splats in Section 4 and regarding our shape program designs in Section 5.

In addition, we provide a video showing the reconstruction progress of our approach through genetic iterations for scenes of the ScanNet dataset [2], qualitative results for our integration of Gaussian splats, and a demo on shape manipulation with PyTorchGeoNodes and Gaussian splats.

1. Additional Validations

In this section, we first provide more details regarding baseline implementations. Then, we provide additional evaluations of our genetic algorithm for the ScanNet dataset and our synthetic dataset. Finally, we provide additional validation for our integration of Gaussian splats into our pipeline.

1.1. Additional Validation on ScanNet

Tables 2, 3 show quantitative evaluations on ScanNet. We compare 3D shapes of our recovered program parameters to 3D shapes of ground truth program parameters using chamfer distance between the corresponding point clouds.

We confirm that our proposed genetic algorithm outperforms alternative baselines. Direct inference with deep learning from [4] struggles to deal with the domain gap and does not generalize to partial point clouds in the ScanNet dataset. While refinement using our PyTorchGeoNodes still helps to improve results, in most cases, the reconstructed parameters and 3D shapes are not representative of the input scene. While our implementation of coordinate descent performs better, it has similar drawbacks, and fails to provide reasonable reconstruction especially in cases of bad random initialisation. In contrast, we observed substantial improvements when relying on our genetic algorithm



Figure 1. Qualitative result on cabinet. Our search algorithm is able to recover correct parameters for ‘Cabinet’ program that includes different number of shelves, existence of doors, size of cabinet, and legs parameters.

to recover shape parameters. We observe very good performance, except in exceptions such as in the case of sofa legs where our objective function is not well-tailored to handle such small details in the scene.

Qualitative results in Figures 6, 5, 7 show additional examples where our approach is able to reconstruct a variety of program parameters that are geometrically consistent with the input scene. In addition, we provide qualitative result for ‘Cabinet’ class in Figure 1 to show our approach can be extended to other categories. We note that we were not able to provide meaningful quantitative results for this category on ScanNet: 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].

We show in Figure 2 that our reconstructions can also be used for segmenting semantic object parts in partial point clouds. In this application, graph of PyTorchGeoNodes directly assigns points in the point cloud to base primitives in our procedural graphs based on simple chamfer distance.

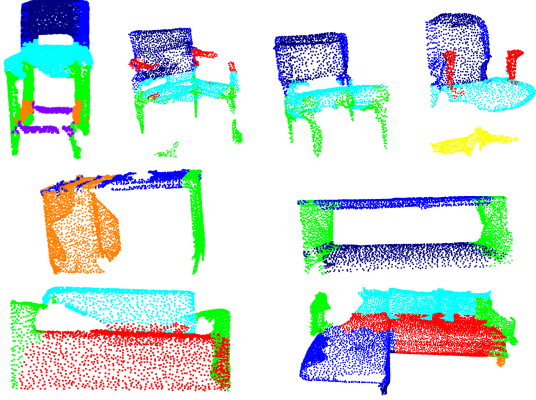


Figure 2. Segmenting semantic parts in partial point clouds of objects based on our reconstructions.

	Initialisation only	Final
SSIM (\uparrow)	0.96	0.99
PSNR (\uparrow)	31.1	36.74
LPIPS (\downarrow)	0.06	0.03

Table 1. Quantitative results on standard appearance metrics of our Gaussian splats integration into PyTorchGeoNodes.

1.2. Validation on Synthetic Data

In addition, we evaluate our search algorithm on synthetic scenes to evaluate performance of our algorithm when there is no presence of occlusions and noise in the data. We generate 300 synthetic scenes that contain complete point clouds of objects corresponding to target parameters. In this experiment, we simply use chamfer distance loss as we do not need to handle occlusions in the scene. We show quantitative results in Tables 4, 5 for 'Cabinet' and 'Chair'. We omit 'Sofa', and 'Table' categories from the supplementary where we observed similar behavior. Similarly to our experiments on ScanNet, we confirm that our genetic algorithm performs extremely well, and we demonstrate advantage of adding gradient descent based on our PyTorchGeoNodes. However, we also observe significant boost in performance compared to our ScanNet experiments. This is a clear indicator that introducing novel objective terms into the search could further improve the reconstruction results.

1.3. Validation of Gaussian Splats Integration

In Table 1 we show quantitative evaluations of our Gaussian splats integration into PyTorchGeoNodes using standard metrics and we mask pixel locations that do not belong to target objects. For the quantitative evaluation, we randomly select 20 views for training and 10 for validation per scene. We selected 10 samples among different categories for this ablation. We choose samples with masks of



Figure 3. Qualitative results for our Gaussian splats integration into PyTorchGeoNodes. We show more results in the supplementary video. We point to the legs of the chair and table. In the input views, there are self-occlusions in case of chair, or severe occlusions from other object. Our computational-graph-aware is able to deal with such situations and we reconstruct appearance of these parts of objects accurately.



Figure 4. Vanilla Gaussian splatting on target objects does not perform well because of occlusions, lack of views of the target object, and noisy observations. In contrast, our integration of Gaussian splatting and PyTorchGeoNodes results in reconstructions of much higher quality, as Gaussians are constrained by the corresponding procedural model.

satisfying quality and accurate shape parameters quality for this experiment for fair evaluation as Gaussian splatting is very sensitive to noise, especially given that we only use 30 views per scene for optimization. We observe that already after initialization the metrics are relatively high and the numbers are further improved after our optimization procedure. In Figure 3 we show qualitative results, and in Figure 4 we compare our results to 'vanilla' Gaussian splatting that is not constrained by our procedural model.

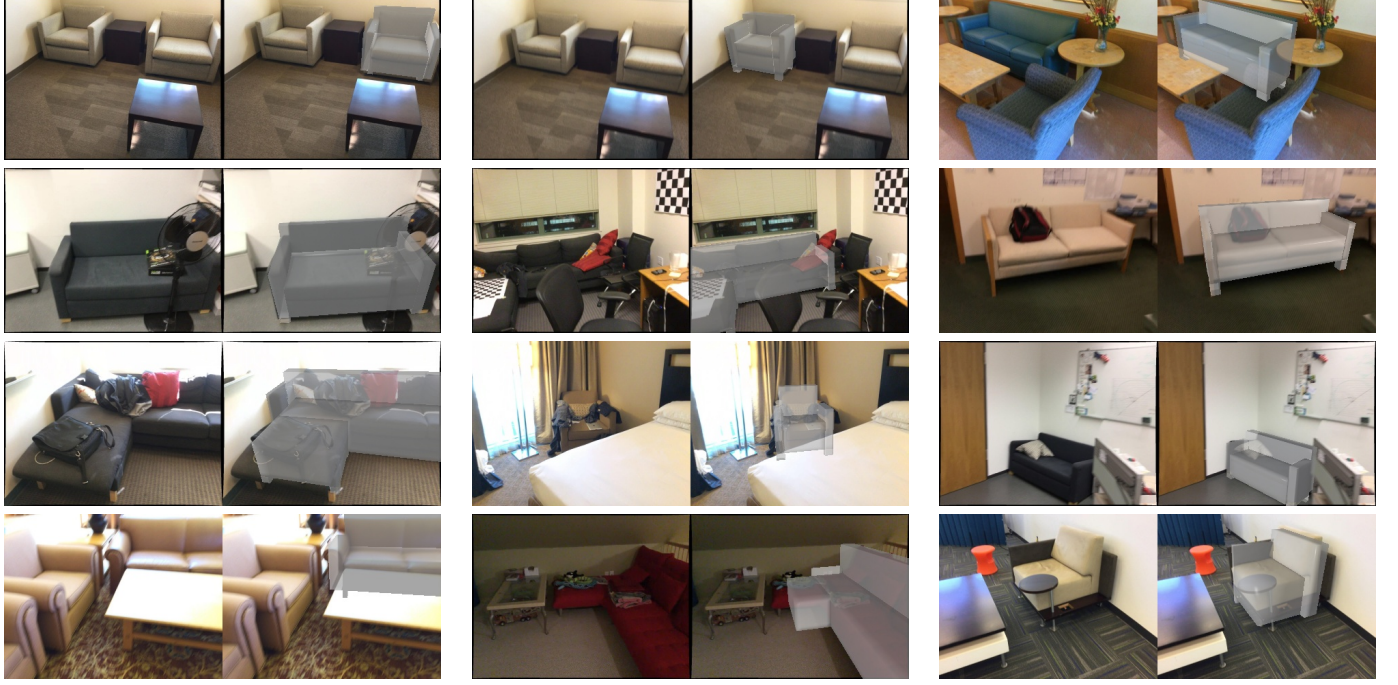


Figure 5. Qualitative results on sofas. For each pair, the left image shows one of the input views. The right images in pairs show our projections of recovered shape parameters. Our results are accurate, dimensions of sofa vary greatly in our validation set, yet we are able to reconstruct these measurements accurately. In addition, we accurately model existence and measurements of armrests, existence and measurements of L-extensions.

Parameter		GeoCode [4] w/o refinement	GeoCode [4]	CD w/o refinement	CD	Genetic w/o refinement	Genetic
Mean Absolute Difference to Ground Truth (↓)							
Continuous Parameters	Width	0.5	0.27	0.18	0.15	0.15	0.12
	Height	0.12	0.08	0.08	0.07	0.08	0.06
	Depth	0.07	0.06	0.12	0.07	0.11	0.07
	Back Height	0.21	0.12	0.12	0.05	0.08	0.06
	Back Depth	0.07	0.08	0.07	0.06	0.07	0.07
	Back Over-Width Scale	0.36	0.35	0.39	0.34	0.39	0.38
	L Depth	0.17	0.2	0.13	0.26	0.11	0.11
	L Width	0.07	0.05	0.09	0.07	0.09	0.05
	Arm Width	0.09	0.08	0.06	0.05	0.06	0.04
	Arm Depth	0.16	0.14	0.04	0.06	0.04	0.06
	Arm Height	0.19	0.23	0.23	0.17	0.18	0.14
	Leg Size	0.03	0.04	0.04	0.04	0.04	0.04
	Leg Height	0.06	0.06	0.04	0.03	0.04	0.03
Classification Accuracy (↑)							
Discrete P.	Has Back	0.4	0.4	0.77	0.96	1.0	1.0
	Is L-Shaped	0.56	0.56	0.77	0.72	1.0	1.0
	Flip L Around Y	0.6	0.6	0.9	0.7	1.0	1.0
	Has Left Arm	0.28	0.28	0.74	0.75	0.95	0.98
	Has Right Arm	0.26	0.26	0.67	0.75	0.95	0.98
	Has Legs	0.46	0.46	0.37	0.47	0.46	0.46

Table 2. Quantitative results on the sofa category.

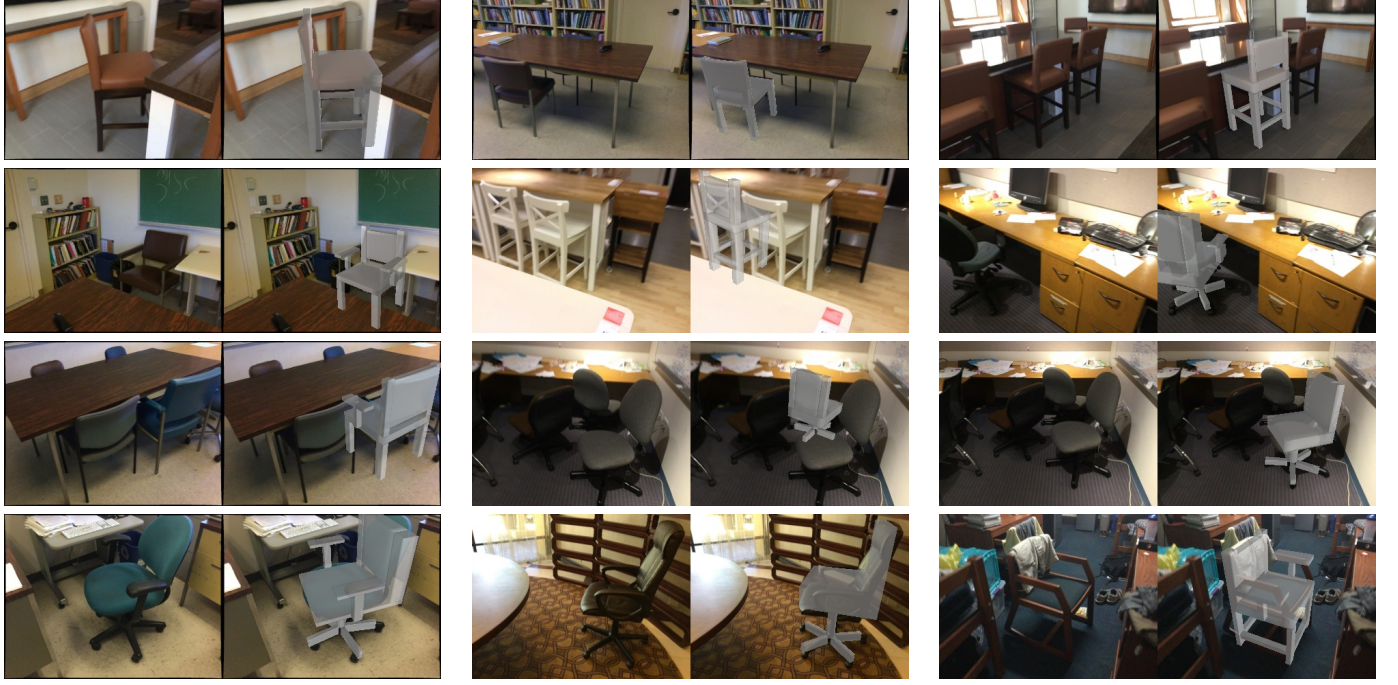


Figure 6. Qualitative results on chairs. For each pair, the left image shows one of the input views. The right images in pairs show our projections of recovered shape parameters. They are accurate, we accurately model thickness of legs, existence position and measurements of leg supports, existence, rotations and measurements of star-shape legs, and existence and measurements of armrests.

Parameter		GeoCode [4] w/o refinement	GeoCode [4]	CD w/o refinement	CD	Genetic w/o refinement	Genetic
Mean Absolute Difference to Ground Truth (\downarrow)							
Continuous Parameters	Seat Width	0.24	0.08	0.06	0.03	0.06	0.03
	Seat Height	0.13	0.1	0.1	0.05	0.08	0.04
	Seat Thickness	0.05	0.04	0.03	0.04	0.03	0.03
	Seat Depth	0.14	0.05	0.07	0.06	0.06	0.05
	Backrest Scale	0.39	0.29	0.22	0.27	0.12	0.09
	Back Height	0.1	0.06	0.05	0.04	0.05	0.03
	Back Thickness	0.02	0.03	0.01	0.02	0.01	0.01
	Backrest Offset Scale	0.33	0.34	0.32	0.3	0.29	0.26
	Legs Size	0.03	0.02	0.03	0.02	0.03	0.01
	Bottom Size Scale	0.37	0.37	0.15	0.38	0.24	0.1
	Bottom Thickness	0.02	0.02	0.03	0.01	0.02	0.02
	Middle Offset 2	0.11	0.12	0.13	0.14	0.14	0.05
	Middle Offset 1	0.13	0.11	0.15	0.14	0.15	0.05
	Middle Support Thickness	0.01	0.01	0.01	0.01	0.01	0.01
	Star Rotation	0.36	0.36	0.19	0.42	0.38	0.1
	Arm Height	0.06	0.06	0.06	0.05	0.06	0.03
	Arm Depth Scale	0.11	0.11	0.1	0.09	0.1	0.05
	Arm Width	0.03	0.03	0.02	0.02	0.02	0.03
	Arm Thickness	0.01	0.01	0.01	0.01	0.01	0.02
Classification Accuracy (\uparrow)							
Discrete P.	Has Back	0.61	0.6	0.67	0.67	1.0	1.0
	Legs Type	0.63	0.62	0.89	0.96	0.94	0.94
	Has Middle Support	0.53	0.55	0.91	0.73	0.91	0.91
	Has Arms	0.44	0.44	0.56	0.69	0.92	0.94

Table 3. Quantitative results on the chair category.

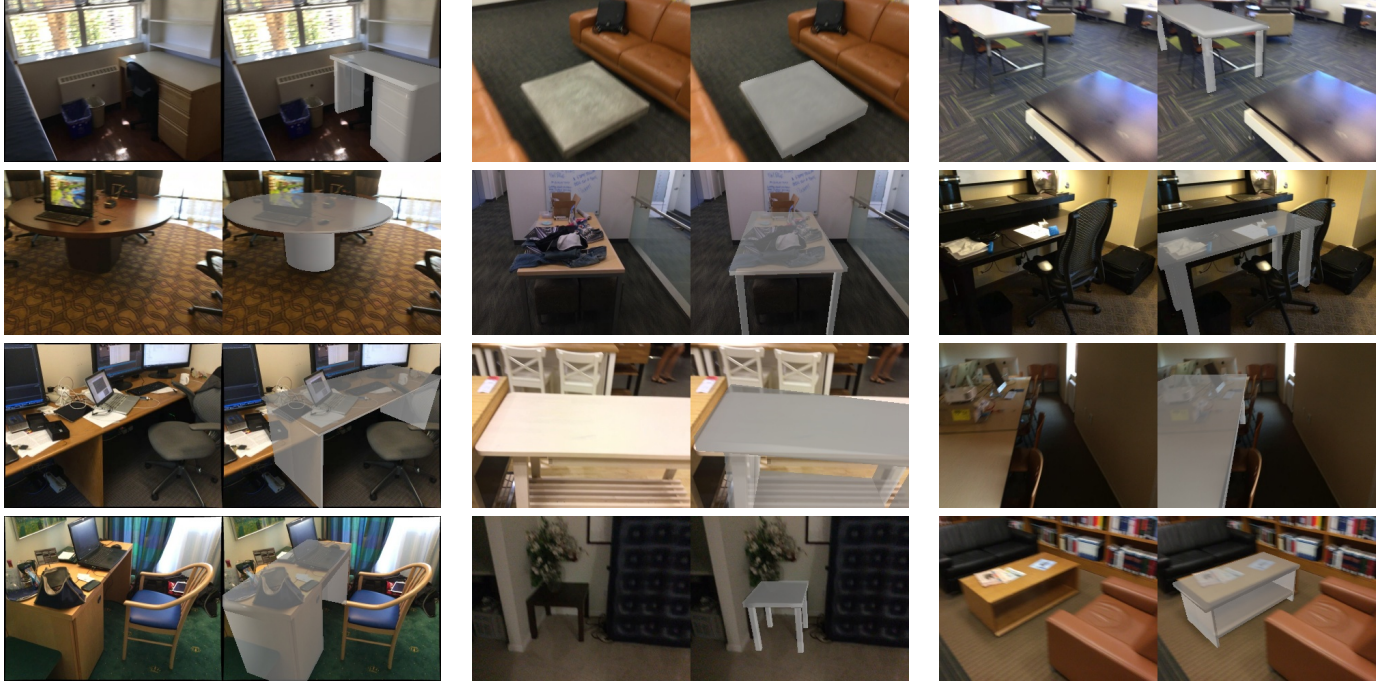


Figure 7. Qualitative results on tables. For each pair, the left image shows one of the input views. The right images in pairs show our projections of recovered shape parameters. Our recovered shape parameters are accurate, dimensions of table vary greatly in our validation set, yet we are able to reconstruct these measurements accurately. In addition, we accurately model existence and position of middle support, existence and measurements of internal cabinet, and shape of the top board.

	Parameter	GeoCode [4]	GeoCode [4]	CD	CD	Genetic	Genetic
		w\o refinement		w\o refinement		w\o refinement	
Continuous Parameters		Mean Absolute Difference to Ground Truth (↓)					
	Height	0.05	0.0	0.05	0.01	0.02	0.0
	Width	0.04	0.01	0.05	0.01	0.02	0.0
	Depth	0.02	0.02	0.05	0.0	0.03	0.0
	Board Thickness	0.0	0.0	0.02	0.0	0.02	0.0
	Dividing Board Thickness	0.0	0.0	0.01	0.0	0.01	0.0
	Leg Width	0.01	0.0	0.02	0.01	0.02	0.0
	Leg Height	0.01	0.01	0.02	0.02	0.02	0.0
	Leg Depth	0.01	0.01	0.02	0.02	0.02	0.01
Discrete Parameters		Classification Accuracy (↑)					
	Has Drawers	1.0	1.0	0.87	0.83	0.97	0.97
	Number of Dividing Boards	1.0	1.0	1.0	1.0	1.0	1.0
	Has Back	1.0	1.0	0.67	0.67	1.0	0.92
	Has Legs	1.0	1.0	1.0	0.57	0.97	1.0
	Has Drawers	1.0	1.0	0.87	0.83	0.97	0.97
	Number of Dividing Boards	1.0	1.0	1.0	1.0	1.0	1.0
	Has Back	1.0	1.0	0.67	0.67	1.0	1.0
	Has Legs	1.0	1.0	1.0	0.57	0.97	1.0

Table 4. Quantitative results on our synthetic dataset for the cabinet category. GeoCode [4] was overfitted to the data from the same distribution and therefore it reaches very high performance but it still benefits from additional refinement enabled by our PyTorchGeoNodes. Our genetic algorithm outperforms coordinate descent and performs comparably to the GeoCode baseline.

		GeoCode [4]	GeoCode [4]	CD	CD	Genetic	Genetic
		w/o refinement	w/o refinement	w/o refinement	w/o refinement	w/o refinement	w/o refinement
Mean Absolute Difference to Ground Truth (\downarrow)							
Continuous Parameters	Seat Width	0.04	0.0	0.06	0.01	0.05	0.0
	Seat Height	0.01	0.0	0.06	0.02	0.03	0.0
	Seat Thickness	0.0	0.0	0.04	0.0	0.04	0.0
	Seat Depth	0.01	0.01	0.05	0.01	0.04	0.01
	Backrest Scale	0.03	0.28	0.18	0.12	0.19	0.03
	Back Height	0.01	0.03	0.05	0.03	0.04	0.0
	Back Thickness	0.0	0.01	0.02	0.0	0.02	0.0
	Backrest Offset Scale	0.04	0.11	0.16	0.16	0.14	0.07
	Legs Size	0.01	0.01	0.02	0.0	0.02	0.0
	Bottom Size Scale	0.13	0.1	0.18	0.32	0.27	0.17
	Bottom Thickness	0.0	0.0	0.01	0.01	0.01	0.0
	Middle Offset 2	0.06	0.05	0.08	0.12	0.1	0.05
	Middle Offset 1	0.02	0.02	0.24	0.18	0.16	0.06
	Middle Support Thickness	0.0	0.0	0.01	0.01	0.01	0.0
	Star Rotation	0.07	0.12	0.23	0.25	0.21	0.27
	Arm Height	0.0	0.0	0.03	0.01	0.03	0.0
	Arm Depth Scale	0.02	0.05	0.08	0.01	0.08	0.01
	Arm Width	0.0	0.01	0.02	0.0	0.02	0.0
	Arm Thickness	0.0	0.0	0.01	0.01	0.01	0.0
Classification Accuracy (\uparrow)							
Discrete P.	Has Back	1.0	1.0	0.9	0.93	1.0	1.0
	Legs Type	1.0	1.0	1.0	1.0	1.0	1.0
	Has Middle Support	1.0	1.0	0.67	0.56	0.89	1.0
	Has Arms	1.0	1.0	0.93	0.97	1.0	1.0

Table 5. Quantitative results on our synthetic dataset for the chair category. GeoCode [4] was overfitted to the data from the same distribution and therefore it reaches very high performance but it still benefits from additional refinement enabled by our PyTorchGeoNodes. Our genetic algorithm outperforms coordinate descent and performs comparably to the GeoCode baseline.

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. Such design enables flow of gradients as we demonstrate in Figure 8. Note that in our experiments we only consider computational graphs that have one output but this is not a limitation of our implementation.

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 re-computed during every forward pass.

3. Search Algorithm – Implementation Details

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

Discretizing continuous parameters. Our genetic algorithm, as well as the coordinate descent baseline, rely on discretization of values for continuous parameters during the search. For a given valid range of a parameter, we generate discrete values in linear steps of 0.2. 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 90° .

Search details. As discussed in the main paper, our genetic algorithm depends on several different hyperparameters. Based on empirical observations from our synthetic experiments, we use the following configuration:

- Total number of generations is set to 50;
- Size of initially randomly generated individuals is 500. We take 64 best performing individuals and maintain this population through iterations;
- At every iteration we generate 128 offsprings by randomly selecting pairs of parents and then randomly selecting values for each of the parameters from this pair.
- Mutation rate is set to $P_m = 0.9$ and linearly decayed to 0.1 through generations, and the random noise added for continuous values is $\sigma = 0.05$.
- To make the search more efficient, we perform refinement every 5, and we randomly select 20 percent of offsprings and perform gradient descent until convergence with Adam optimizer and learning rate 0.01.

4. Integration of Gaussian Nodes – Implementation Details

As explained in the main manuscript, we integrate Gaussian nodes into PyTorchGeoNodes in a straight-forward way, by adjusting the nodes that generate primitive meshes, in our case Cubes and Cylinders, to also generate Gaussian parameters that are fitted to the meshes of corresponding primitives. More precisely, we are inspired by findings from SuGaR [3] in order to ensure that object mesh and Gaussians remain consistent:

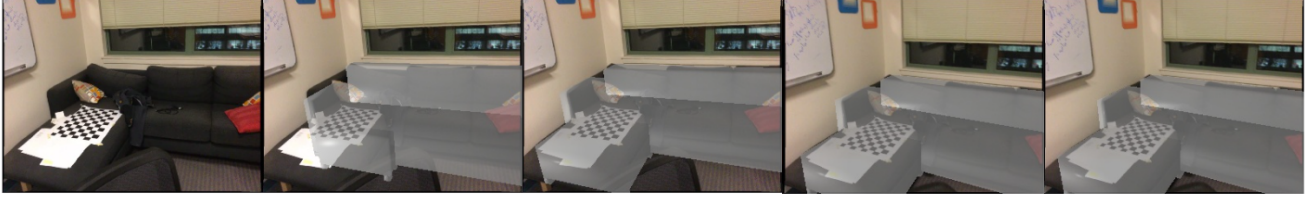


Figure 8. **Gradient-based optimization of continuous parameters of a shape program for the sofa category.** From an initial estimate of the parameters of the object, we can perform gradient descent on the parameters based on a 3D geometric loss term. In contrast to methods that directly optimize the reconstructed mesh, PyTorchGeoNodes allows optimization in the parameter space which has several benefits. From the resulting shapes in this example, it is observable that individual parameters can be scaled independently targeting only specific parts of the shape geometry while preserving the compactness of the 3D shape at the same time.

- Means are not optimized directly, we compute means explicitly from triangles of object mesh, and we use 4 means per face in barycentric coordinate system relative to the individual faces. Instead, we optimize offsets of vertices of the object mesh.
- We keep Gaussians flat by freezing their thickness to 10^{-3} , and only optimize the remaining two scales. They are initialized to be isotropic and are set to 0.5 of maximum side length of corresponding triangles. We optimize offsets to these scales.
- Gaussian rotations are initialized to be aligned with the object mesh and we only allow optimization of offset to the in-plane rotation.
- We initialize colors from the closest points in the point cloud of the corresponding object.
- We also optimize opacity which is initially set to 0.7 in our experiments.

We train the parameters for 1000 iterations. We use the Adam optimizer, and as is the case for different implementations of Gaussian splats, we observed that using different learning rates for different parameters improves convergence: for mesh vertex offset we use learning rate 10^{-4} , for scales offsets 5^{-3} , for colors 10^{-2} , and for opacity 10^{-2} .

5. Shape Program Designs for Validation

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

‘Chair’ consists of 23 parameters, 19 continuous, and 4 boolean parameters:

- ‘Legs Type’ is a boolean parameter that determines whether a chair has four legs or one star-shaped 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-1’ is a continuous parameter that controls the height offset of left and right leg support relative to the seat height. The valid range of values is in $[0, 0.5]$;
- ‘Support Offset-2’ is a continuous parameter that controls the height offset of back and front 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 surfaces 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 $[0.7, 1.0]$;
- ‘Star Rotation’ is a continuous parameter that controls the rotation of the bottom component of one-legged chairs normals to range $[0, 1]$ in radians. The valid range of values is in $[0.0, 1.00]$;
- ‘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 19 parameters, 13 continuous parameters and 6 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];
- 'Leg Height' is a continuous parameter that controls the height of legs in meters. The valid range is in [0.01, 0.2];
- '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];
- 'Back Over-Width Scale' determines aspect ratio between width of backrest and seat, and relative to the armrest width. The valid range is in [0.0, 1.0];
- '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 15 parameters, 11 continuous, and 4 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 leg in the middle, 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].
- 'Has Cabinet Leg' is a boolean parameter that determines if a four-legged table should have an integrated cabinet in place of legs on one side of the table.
- 'Legs Scale X' is a continuous parameter that scales leg of the table relatively to the table width. The valid range is in [0.01, 0.1].
- 'Legs Scale Y' is a continuous parameter that scales leg of the table relatively to the table depth. The valid range is in [0.01, 0.5].
- 'Legs Offset X' is a continuous parameter that offsets leg of the table relatively to the table width. The valid range is in [0.0, 1.0].
- 'Legs Offset Y' is a continuous parameter that offsets leg of the table relatively to the table depth. The valid range is in [0.0, 1.0].

'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.

References

- [1] Stefan Ainetter, Sinisa Stekovic, Friedrich Fraundorfer, and Vincent Lepetit. HOC-Search: Efficient CAD Model and Pose Retrieval from RGB-D Scans. *International Conference on 3D Vision*, 2024. [1](#)
- [2] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *Conference on Computer Vision and Pattern Recognition*, 2017. [1](#)
- [3] Antoine Guédon and Vincent Lepetit. SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5354–5363, 2024. [7](#)
- [4] Ofek Pearl, Itai Lang, Yuhua Hu, Raymond A Yeh, and Rana Hanocka. GeoCode: Interpretable Shape Programs. *arXiv Preprint*, 2022. [1](#), [3](#), [4](#), [5](#), [6](#)