EUROGRAPHICS 2025 / A. Bousseau and A. Dai
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), *Number 2*

Approximating Procedural Models of 3D Shapes with Neural Networks: Supplementary Material

Ishtiaque Hossain[1] I-Chao Shen[2] Oliver van Kaick[1]

[1] Carleton University, Canada
[2] The University of Tokyo, Japan

1. Details of the primitive-based procedural model

We developed a procedural model which creates four classes of 3D man-made furniture shapes from vectors of parameters. The classes are beds, chairs, shelves and tables. In this section, we provide the details of the procedural model.

The procedural model assembles primitive shapes with transformations applied to them, in order to create complex shapes. In our procedural model, we use the cube primitive and the cylinder primitive. The transformations applied to the primitives, i.e., scaling, rotation, and translation, dictate the size, orientation and location of the primitives, respectively. Both the selection of the primitives and the transformations are derived from the parameter vector passed to the procedure.

As mentioned before, the parameter vector is a mix of different types of individual parameters. Parameters can be continuous values, in which case, the corresponding parameter is normalized. Discrete parameters can be binary, integer intervals, or finite sets.

The final shape is in triangle mesh format. In some cases, additional operations are performed on the primitives which changes their geometry. For instance, edges can be bevelled in order to give certain parts of the shapes more realistic look, or faces can be extruded in order to simplify the creation of parts. Physic-based simulation is used as well, to create parts such as pillows.

To facilitate the creation of primitives and performing various operations on them, we use the BMesh module (Blender's internal mesh editing API) available in Python. More details on the module can be found at `https://docs.blender.org/api/current/bmesh.html`. In the following sections, we provide details about each class of shapes with examples.

1.1. Bed

The procedure for creating the bed shapes has 67 lines of code and accepts 7 parameters.

- Width to height ratio: Controls the aspect ratio of the bed, the depth is fixed.
- Leg height: Controls the height of the legs.
- Headboard height: Controls the height of the headboard.
- Frontboard height: Controls the height of the frontboard.
- Mattress height: Controls the height of the mattress.
- Leg type: Controls which type of leg the bed has, any of {basic, solid, box}.
- Number of pillows: Controls the number of pillows on the bed, can be in the range [0, 2].

Figure 1 shows examples of bed shapes where the effects of each parameter on the output shapes is illustrated.

1.2. Chair

The procedure for creating the chair shapes has 114 lines of code and accepts 6 parameters.

- Width to height ratio: Controls the aspect ratio of the chair.
- Depth: Controls the depth of the chair.
- Leg height: Controls the height of the legs.
- Leg type: Controls which type of leg the chair has, any of {basic, round, support, office}.
- Arm type: Controls which type of arm the chair has, any of {none, basic, solid, office}.
- Back type: Controls which type of back the chair has, any of {basic, horizontal bars, vertical bars, office}.

Figure 2 shows examples of chair shapes where the effects of each parameter on the output shapes is illustrated.

1.3. Shelf

The procedure for creating the shelf shapes has 51 lines of code and accepts 8 parameters.

- Width to height ratio: Controls the aspect ratio of the shelf.
- Depth: Controls the depth of the shelf.
- Leg height: Controls the height of the legs.
- Number of rows: Controls the number of rows in the shelf, can be in the range [1, 5].
- Number of columns: Controls the number of columns in the shelf, can be in the range [1, 5].
- Filled back: Binary parameter, indicating whether the shelf has a solid or open back.

- Filled sides: Binary parameter, indicating whether the shelf has solid or open sides.
- Filled columns: Binary parameter, indicating whether adjacent columns have solid or open separation.

Figure 3 shows examples of shelf shapes where the effects of each parameter on the output shapes is illustrated.

1.4. Table

The procedure for creating the table shapes has 60 lines of code and accepts 6 parameters.

- Width to height ratio: Controls the aspect ratio of the table.
- Depth: Controls the depth of the table.
- Top thickness: Controls the thickness of the tabletop.
- Leg thickness: Controls how thick the table legs are.
- Rounded top: Binary parameter, indicating whether the top is rounded or rectangular.
- Leg type: Controls which type of leg the table has, can be any of {basic, support, round, split, square, solid}.

Figure 4 shows examples of table shapes where the effects of each parameter on the output shapes is illustrated.

2. Details of the node graph-based procedural model

The procedural model for creating sofa shapes is a third-party Blender-based model where shapes are created using a node graph. The node graph is an interconnected network of nodes, where each node represents an operation. Nodes can have input and output. Among other types of nodes, the node graph heavily utilizes geometry nodes, which are responsible for changing shape geometry. The inputs to the node graph are numeric values representing the parameters to the procedural model and the node graph itself represents the procedure. The output of the node graph is the final shape, represented in triangle mesh format. The node graph in this particular case has 19 adjustable parameters. However, allowing all of them to change freely often leads to unrealistic strange shapes. To remedy this issue, we set 7 of these parameters to constant values and let the other parameters vary within allowable ranges. Below is a list of these 12 parameters.

- Depth: Depth of the sofa from front to back.
- Legs height: Height of the legs.
- Back/side height: Height of the sofa back and the sides (if any).
- Side thickness: Width of of the sides (if any).
- Cushion thickness: Thickness of the seat cushions.
- Back cushion height: Height of the back cushions.
- Back cushion thickness: Thickness of the back cushions.
- Side cushion thickness: Thickness of the side cushions (if any).
- Side cushion height: Height of the side cushions (if any).
- Number of sides: The number of sides the sofa has. The values are integers in the range [0, 2].
- Number of cushions: The number of seat-back cushion pairs. The values are integers in the range [0, 2].
- Number of side cushions: Number of cushions the sofa has on its sides. The values are integers in the range [0, 2].

Figure 5 shows examples of sofa shapes where the effects of each parameter on the output shapes is illustrated.

Table 1: *Additional information on the shapes generated by the two procedural models.*

Class	Bed	Chair	Shelf	Table	Sofa
Avg # of vertices	1,224	338	170	69	7,244
Avg # of faces	2,436	657	284	121	14,462
# of parts	5	4	5	2	6

Table 2: *Parameter prediction error for each shape category for various configurations of the proposed network.*

		Bed	Chair	Shelf	Table	Sofa
Task 2	scalar	0.02	0.01	0.03	0.05	0.10
	binary	-	-	0.93	1.00	-
	integer	0.56	0.99	0.95	0.99	0.98
Task 3	scalar	0.09	0.05	0.09	0.14	0.21
	binary	-	-	0.83	0.98	-
	integer	0.59	0.94	0.80	0.84	0.81
Task 4	scalar	0.42	0.19	0.20	0.28	0.39
	binary	-	-	0.58	0.64	-
	integer	0.28	0.46	0.23	0.56	0.40

3. Statistics of generated shapes

Table 1 reports some additional information on the shapes that are created by the two procedural models, such as the average number of vertices/faces and the number of parts for each class of shapes.

4. Additional Experimental Results

4.1. Parameter Prediction Error

In this section, we present quantitative results from the experiments where our network was configured to infer parameters for unseen shapes in various settings. Table 2 shows the performance of our network when predicting parameters. Task 2 refers to parameter prediction from voxel-grids directly, Task 3 also refers to parameter prediction from voxel-grids, but using optimization. Finally, Task 4 refers to parameter prediction from silhouette images using optimization. For scalar parameters, we report the average Mean Absolute Error (MAE) which is normalized to 1.0. For both binary and integer parameters, we report the average F1 score.

4.2. Comparison with ShapeAssembly

In this section, we present additional qualitative results from the comparison between our method and ShapeAssembly. Figure 6 shows 30 examples where we select shapes from the ShapeNet dataset and reconstruct them using both ShapeAssembly and our method. It can be seen that, in some cases, ShapeAssembly performs better, while in other cases, our method works better. One advantage of our method is that the reconstruction is always a valid shape, since the final reconstruction is performed using the original procedural model. On the other hand, ShapeAssembly can produce reconstructions with bad geometry.

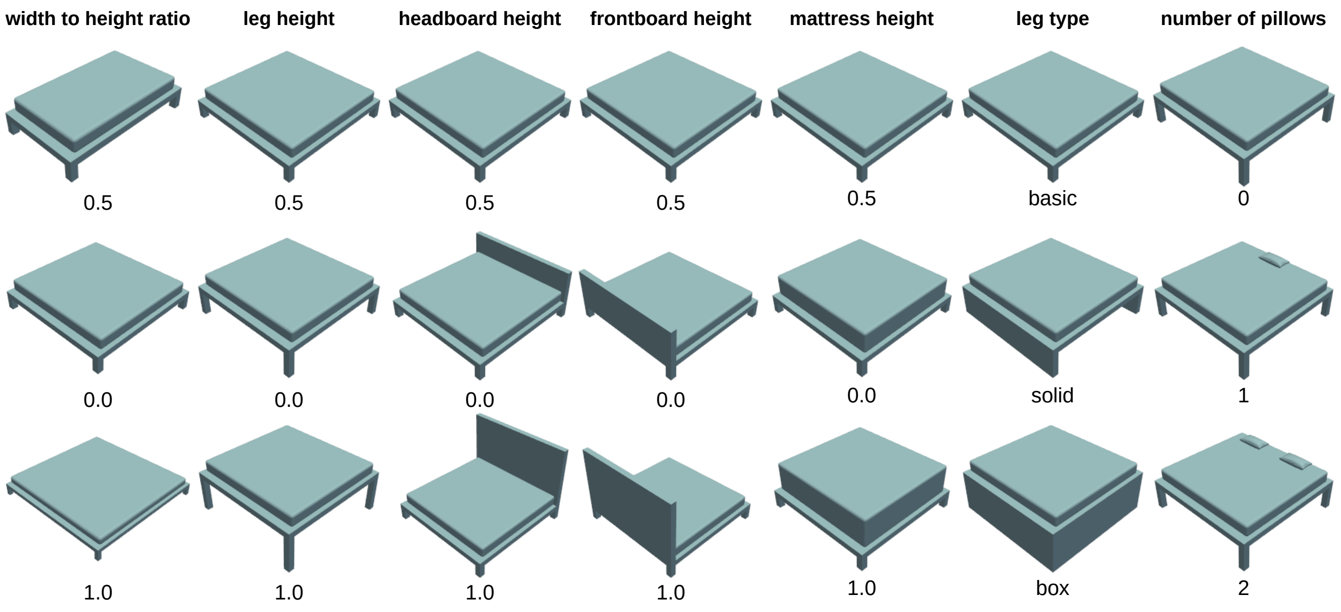

Figure 1: *Examples of bed shapes created by the primitive-based procedural model.*

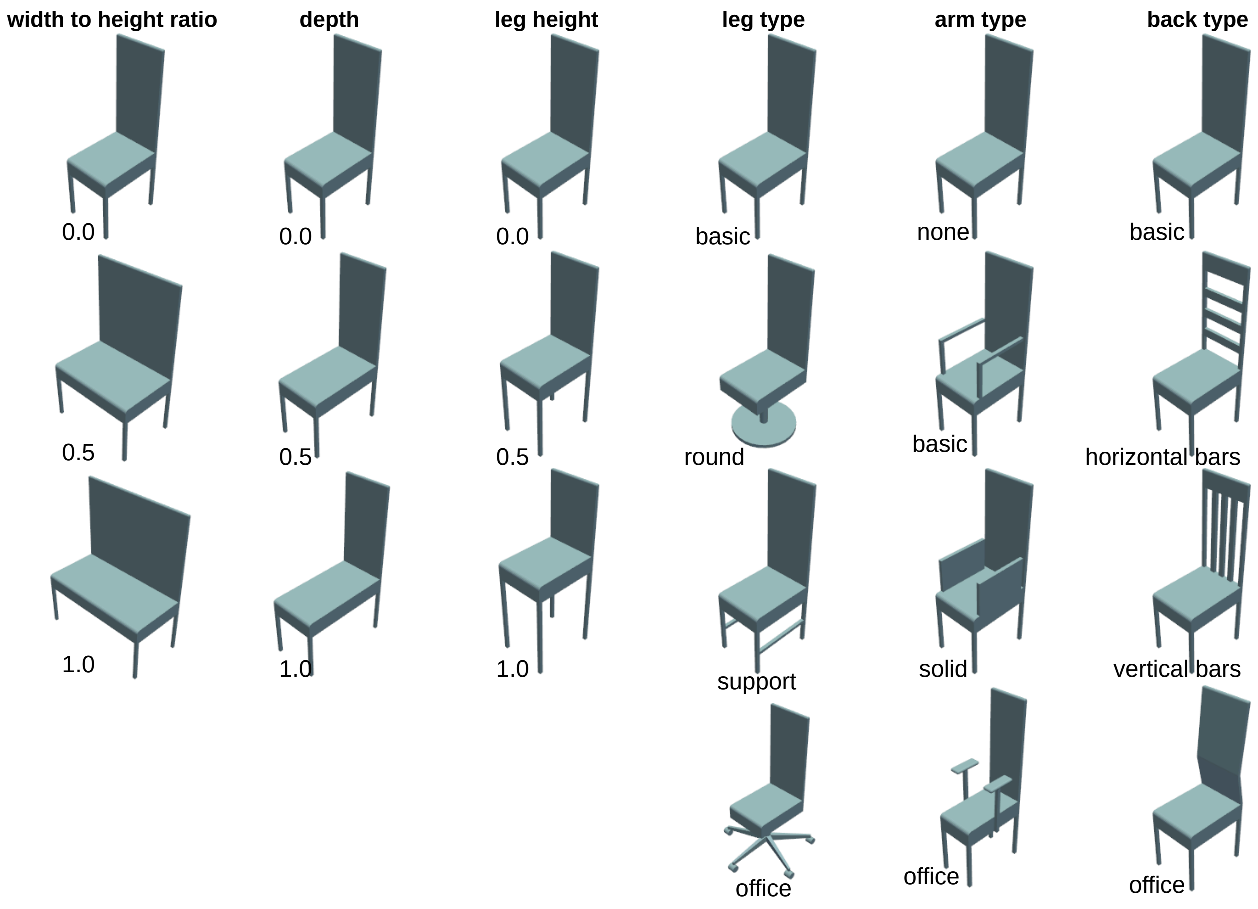

Figure 2: *Examples of chair shapes created by the primitive-based procedural model.*

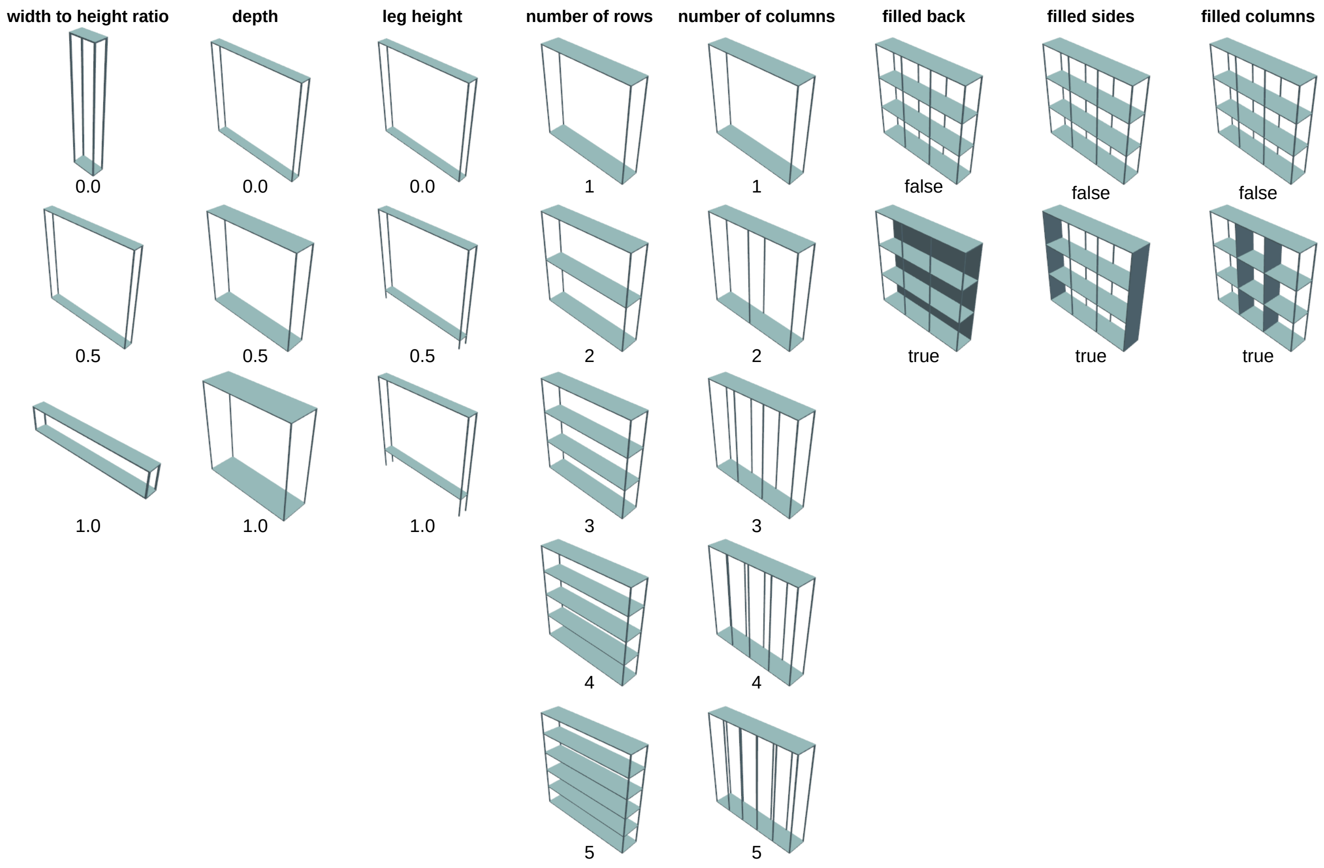

Figure 3: *Examples of shelf shapes created by the primitive-based procedural model.*

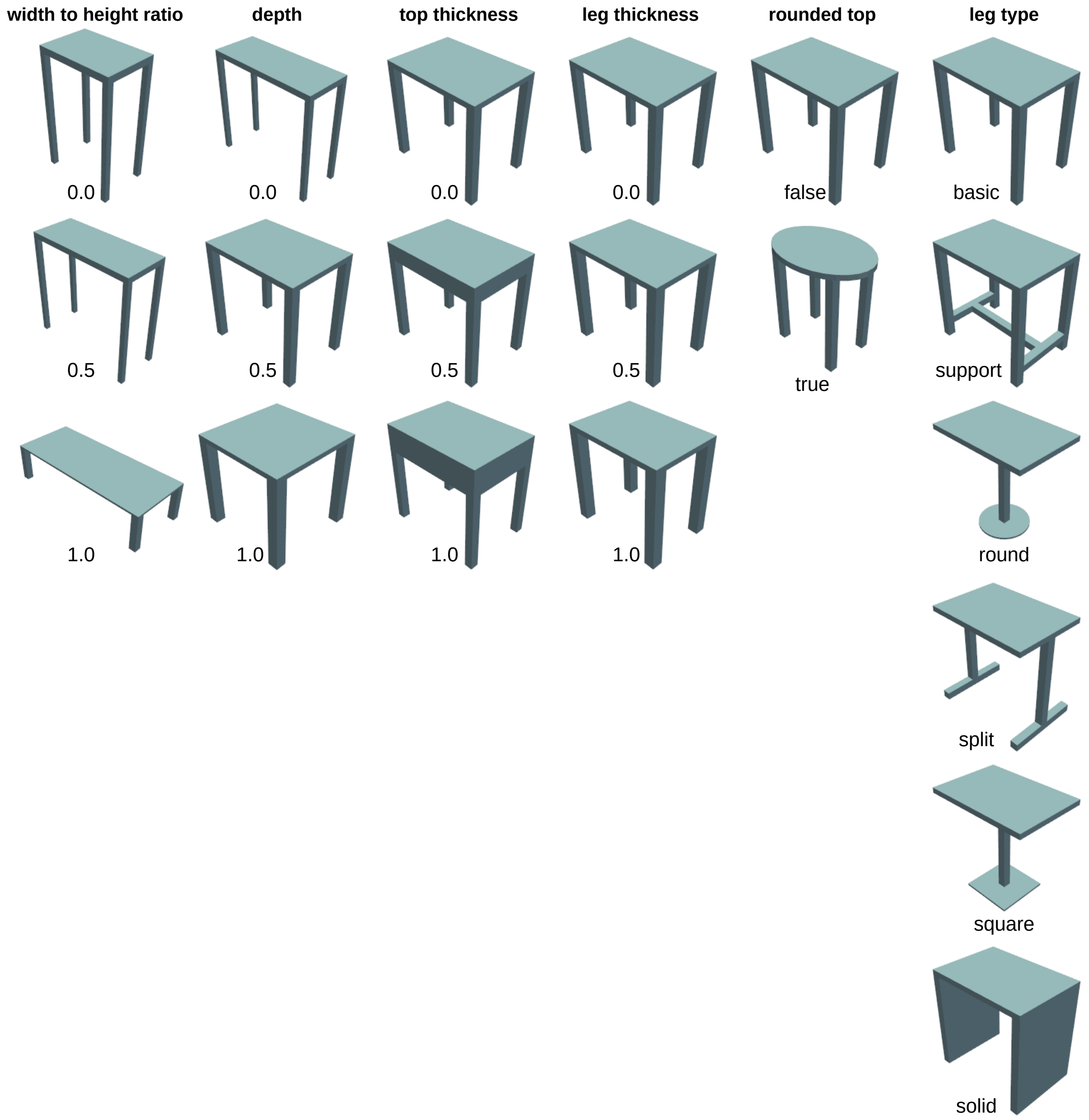

Figure 4: *Examples of table shapes created by the primitive-based procedural model.*

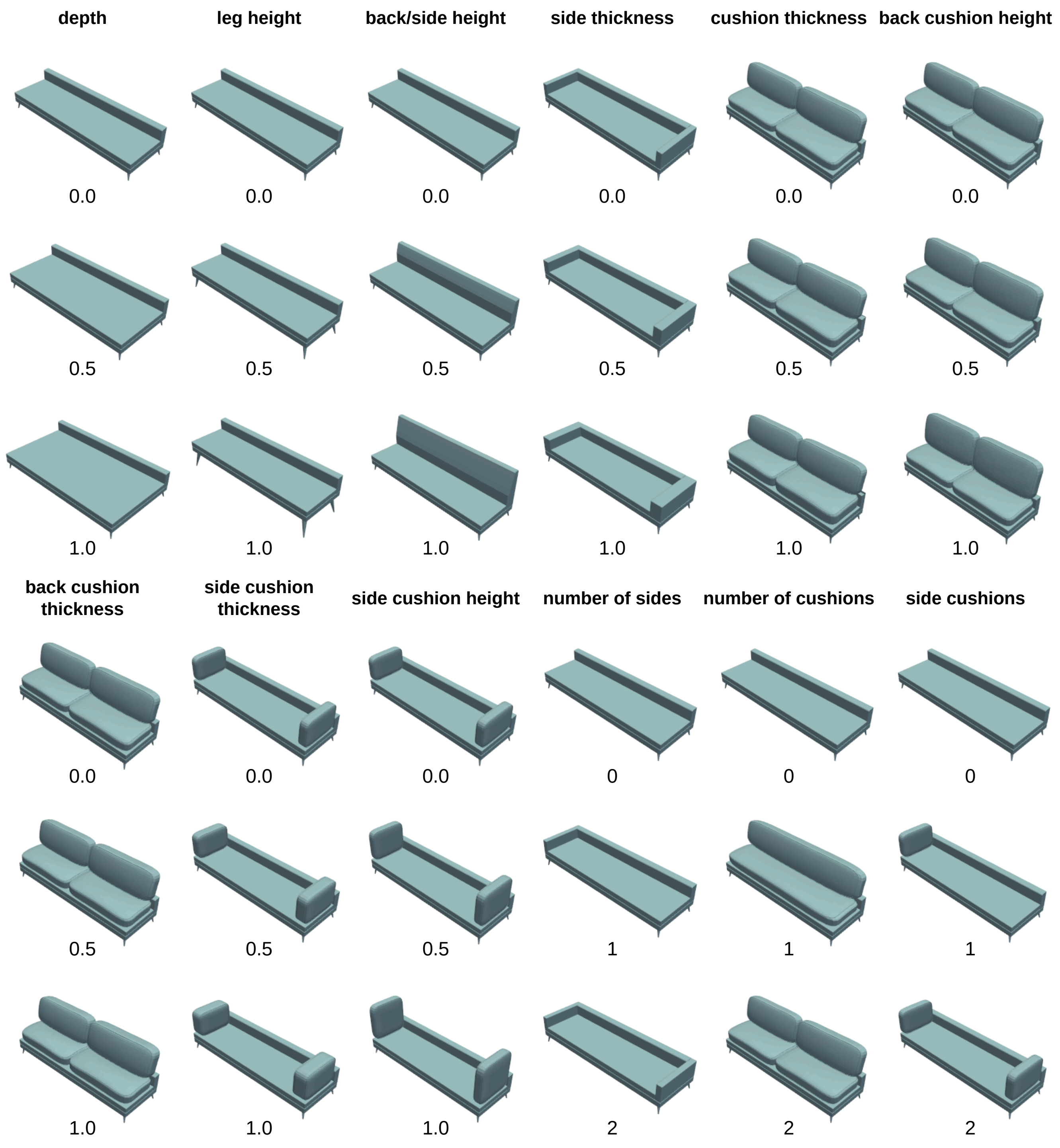

Figure 5: *Examples of sofa shapes created by the node graph-based procedural model.*

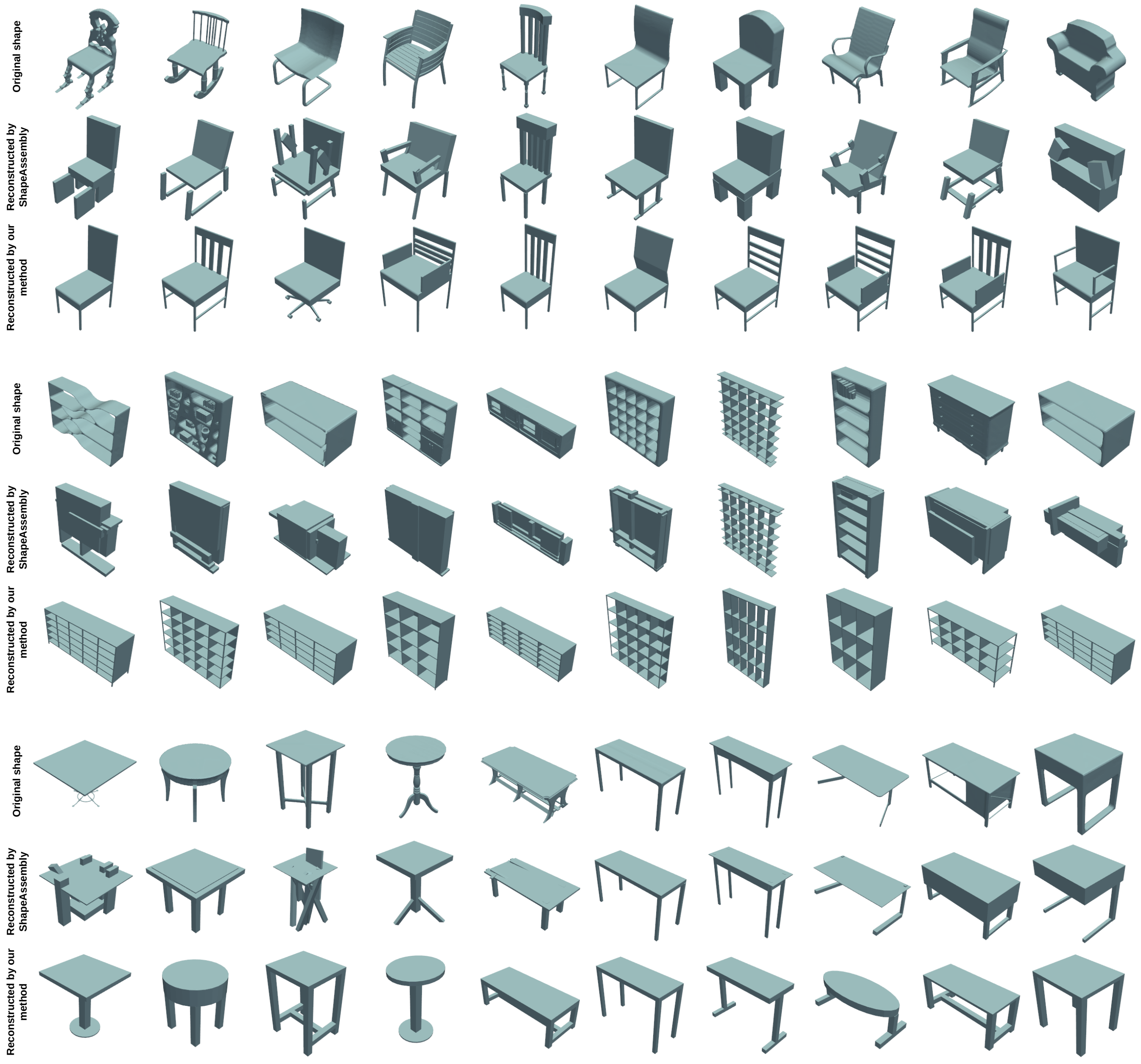
Original shape
Reconstructed by ShapeAssembly
Reconstructed by our method
Original shape
Reconstructed by ShapeAssembly
Reconstructed by our method
Original shape
Reconstructed by ShapeAssembly
Reconstructed by our method

Figure 6: *Comparison between reconstruction by ShapeAssembly and our method on selected shapes.*