

## 715 A Appendix of MeshLLM: LLM-Powered Structured Mesh Code Generation 716 from Point Clouds

### 717 A.1 Datasets

#### 718 A.1.1 The principles of Translation and Bridge Loop

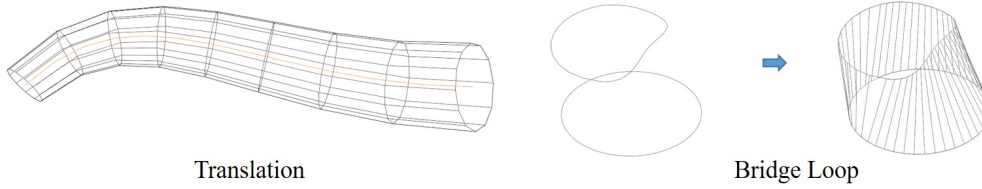


Figure 10: A schematic illustration of the principles of Translation and Bridge Loop. In the Translation module, the wireframe of the resulting mesh is shown as a cross-sectional circle is translated along a yellow trajectory. In the Bridge Loop module, the wireframe of the mesh is constructed by connecting the vertices of two 2D shapes.

719 As illustrated in the figure 10, in the Translation operation, a 2D cross-sectional shape (a circle in this  
720 example) and a 3D trajectory curve must first be defined. The Translation process generates a mesh  
721 by sweeping the 2D shape along the 3D trajectory. During this sweep, the cross-section remains  
722 perpendicular to the tangent direction of the trajectory at all times, and only uniform scaling (either  
723 enlargement or reduction) of the cross-section is permitted.

724 In contrast, the Bridge Loop operation begins with two predefined 2D shapes. By connecting the  
725 corresponding vertices of these two shapes, a mesh can be constructed. This method places no  
726 constraints on the types of 2D shapes used—meaning the two shapes can differ, such as a circle  
727 and an irregular closed shape in this example. Moreover, it imposes no restrictions on the relative  
728 orientations of the shapes. As a result, Bridge Loop overcomes the limitations of Translation, which  
729 requires the cross-section to align with the trajectory’s tangent direction. This enables Bridge Loop  
730 to generate more complex geometries that Translation cannot produce.

#### 731 A.1.2 Part datasets

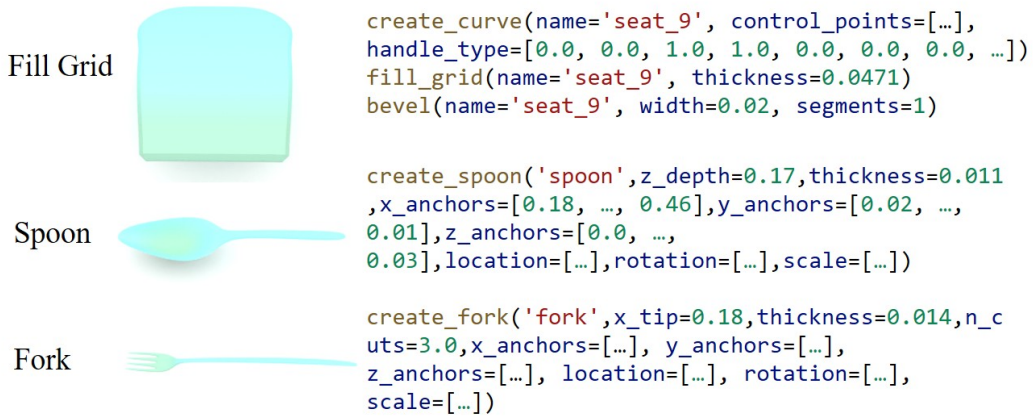


Figure 11: The Fill Grid type, Spoon type and Fork type in basic shape code library

732 For certain shapes that are difficult to represent using the method we defined in Section 3.1, we  
733 introduce three additional categories: the Fill Grid type, Spoon type and Fork type. As illustrated  
734 in the Figure 11. For the Fill Grid type, we first construct a closed 3D shape (as opposed to the  
735 2D cross-sectional shape used in Translation), fill it to form a surface, and then extrude it along its

736 normal direction to generate the final mesh. For the Spoon and Fork type, we draw inspiration from  
 737 the implementation in Infinigen Indoor (19) and design dedicated procedural functions tailored for  
 738 their generation.

739 We present two core functions from our codebase: the complete implementation for creating primitives  
 740 (Figure 22) and the complete implementation for creating curves (Figure 23). The full codebase can  
 741 be found in the supplementary materials.

742 More examples of parts and their corresponding complete code implementations are provided in  
 743 Figures 12, 13, 14, 15, and 16.

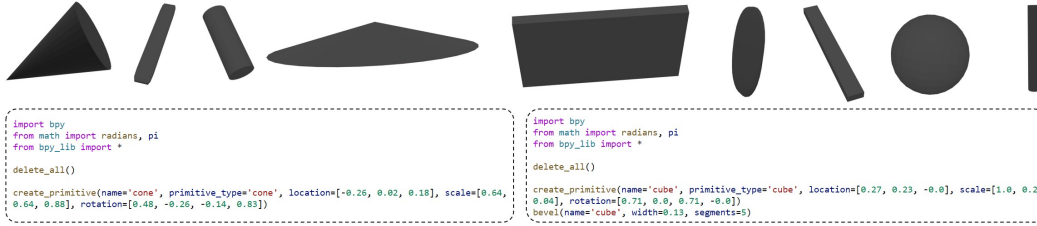


Figure 12: Examples of Primitive and complete code. And the code corresponds to the first two objects shown in the figure.

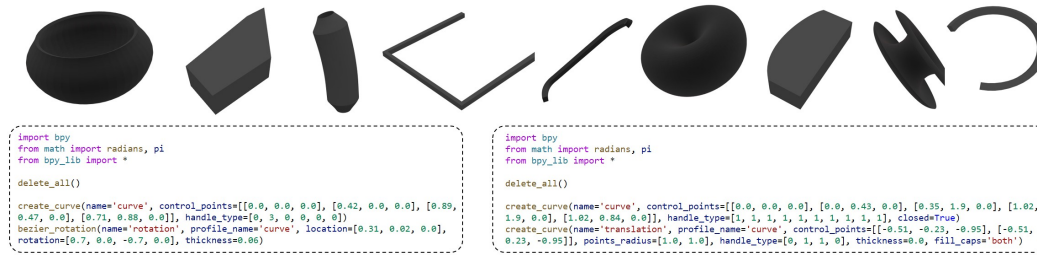


Figure 13: Examples of Translation and complete code. And the code corresponds to the first two objects shown in the figure.

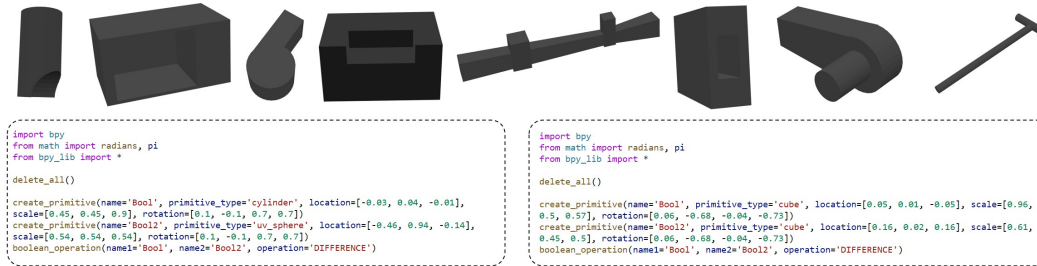


Figure 14: Examples of Boolean and complete code. And the code corresponds to the first two objects shown in the figure.

744 Taking the Primitive type as an example, we describe how to use functions from the basic shape  
 745 code library to generate a synthetic part dataset. We begin by randomly selecting the type of primitive  
 746 to generate (e.g., cube, cylinder, etc.). Next, for each axis, we independently uniform sample  
 747 a value  $x$  from the range  $[-2, 2]$ , and then set the corresponding scale as  $10^x$ . To determine the  
 748 orientation of the shape, we uniformly sample a direction from a unit sphere and a roll angle from  
 749 a uniform distribution. Once the orientation is fixed, we scale the shape uniformly along all three  
 750 axes based on the size of its bounding box. Specifically, we ensure that the longest edge of the  
 751 bounding box lies within the range  $[1, 2]$ . Finally, we assign the shape a random position within the  
 752 3D space such that the entire shape remains within the  $[-1, 1]$  bounds. For other shape types beyond  
 753 Primitive, we follow a similar approach by randomly assigning values to the relevant parameters.

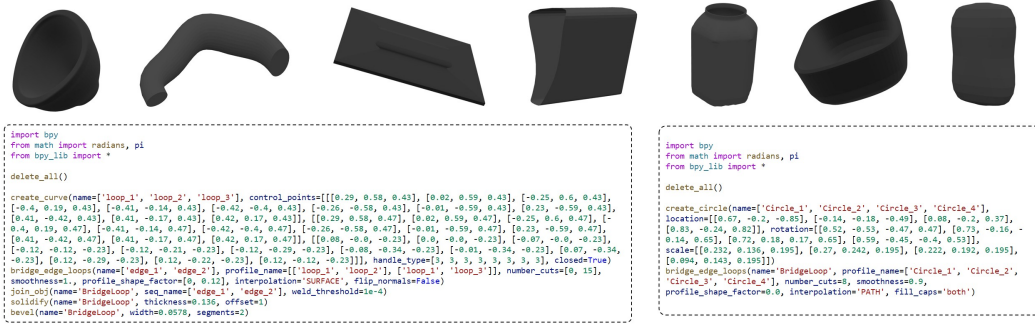


Figure 15: Examples of Bridge Loop and complete code. And the code corresponds to the first two objects shown in the figure.

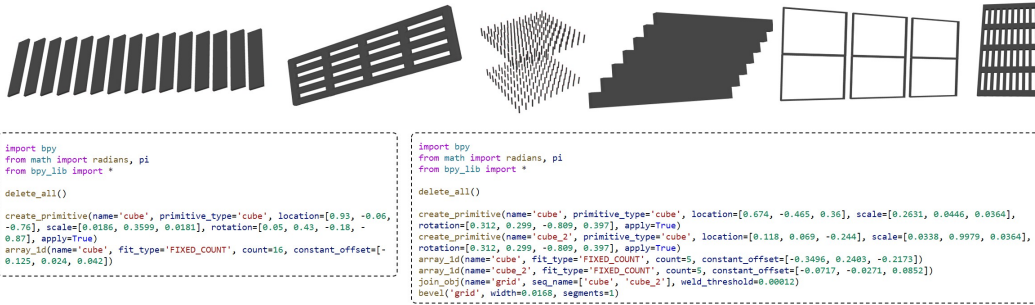


Figure 16: Examples of Array and complete code. And the code corresponds to the first two objects shown in the figure.

### 754 A.1.3 Object datasets

755 For assembling part codes into a complete program, we provide a full example containing the  
756 complete code, as shown in Figure 17. Regarding the ordering strategy used when assembling parts  
757 into a complete object, we adopt a consistent spatial heuristic to determine part sequence. Specifically,  
758 parts are arranged from bottom to top, left to right, and front to back. To implement this, we divide  
759 the 3D space into a  $32 \times 32 \times 32$  grid and assign each part a characteristic grid cell that serves as the  
760 basis for sorting. The characteristic grid cell of a part is defined as follows: among all grid cells that  
761 the part occupies, we first select the one with the smallest  $z$ -coordinate. If multiple candidates share  
762 the same  $z$ -value, we choose the one with the smallest  $x$ -coordinate. If a tie still exists, we select the  
763 one with the smallest  $y$ -coordinate. Parts are then sorted based on the lexicographic order of these  
764 characteristic grid cells, which determines their final sequence within the object.

765 It is important to note that for each object, the prerequisite for successfully constructing its corre-  
766 sponding code lies in the ability of our part-to-code inference model to accurately infer all of its  
767 individual parts. We consider a part to be successfully inferred if the Chamfer Distance (CD) between  
768 the predicted point cloud and the ground truth is below  $5 \times 10^{-3}$ . Therefore, when constructing the  
769 object-code pairs dataset, we only include objects for which **all** constituent parts meet this criterion.  
770 Objects with any part failing to meet this standard are discarded. As a result, the number of success-  
771 fully constructed object-code pairs is smaller than the total number of objects in the original Infinigen  
772 dataset. In fact, the original Infinigen dataset we use contains 1.57 million object instances, from  
773 which we successfully construct 1 million shape-code pairs. For training and evaluation, we split the  
774 full Infinigen dataset into 70% for training, 15% for testing, and 15% for validation. Accordingly,  
775 MESHLLM is trained only on the subset of the shape-code pairs that fall within the training portion  
776 of the Infinigen dataset. In contrast, the baseline models are trained on the full set of objects in the  
777 training split of the original Infinigen dataset. Importantly, all evaluation results for our method and  
778 the baselines are reported on the same test set, i.e., the testing split of the complete Infinigen dataset.

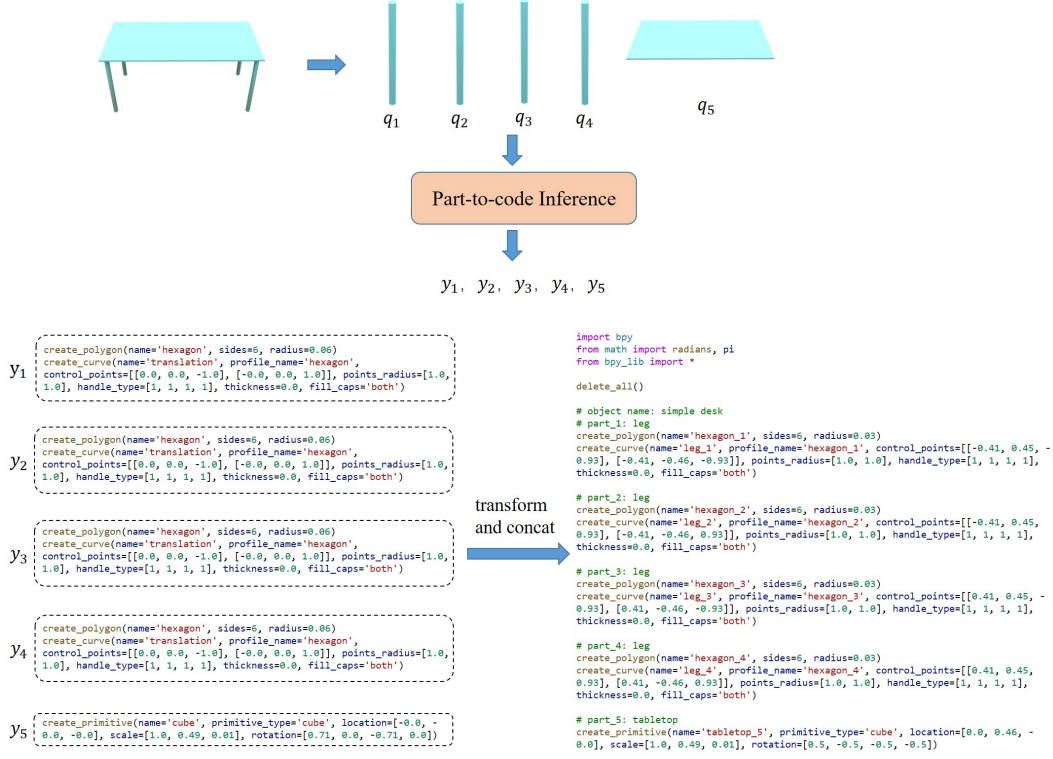


Figure 17: A complete code example of converting part codes into a full object program.

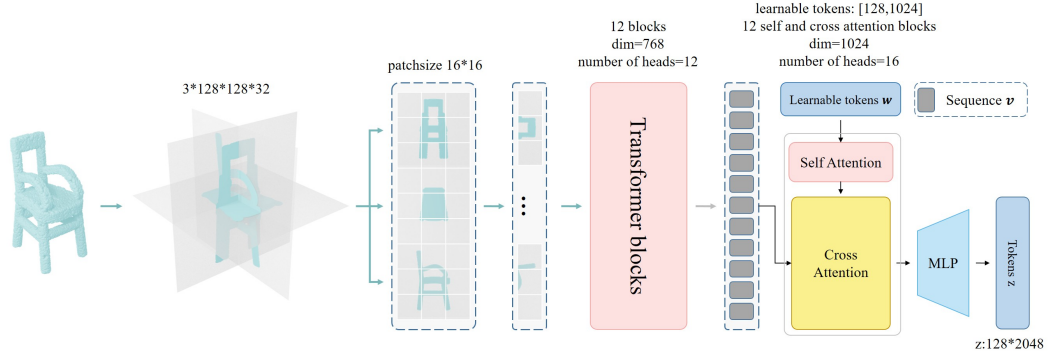


Figure 18: Detailed configuration of the shape tokenizer.

## 779 A.2 Model architecture

780 We explain the detailed structure of the shape tokenizer. As illustrated in the Figure 18, we first  
 781 project the input point cloud of shape  $\mathbb{R}^{n \times 3}$  onto three orthogonal planes to obtain tri-plane features  
 782 with shape  $\mathbb{R}^{3 \times 128 \times 128 \times 32}$ . With a patch size set to  $16 \times 16$ , these tri-plane features are encoded  
 783 into tokens and fed into Transformer blocks, where the resulting representation is mapped to  $v$  and  
 784 used as the key and value ( $K$ ,  $V$ ) inputs. Meanwhile, a set of learnable tokens with shape  $\mathbb{R}^{128 \times 1024}$   
 785 are used as queries in a self and cross attention module. After passing through 12 layers of self and  
 786 cross attention, we obtain output tokens of shape  $\mathbb{R}^{128 \times 1024}$ , which are then projected to the final  
 787 representation of shape  $\mathbb{R}^{128 \times 2048}$  via an MLP.



### 788 A.3 More training details

789 For the part-to-code reconstruction model, we adopt the AdamW optimizer and train it for 20 epochs  
 790 on 64 NVIDIA A100 GPUs for about a week with a batch size of 512, and a learning rate of  $10^{-4}$ .  
 791 We evaluate the model at every epoch and select the checkpoint with the lowest  $L_2$  Chamfer Distance  
 792 (CD) loss. Then we initialize the weights of the object-to-code reconstruction model with the weights  
 793 of the trained part-to-code reconstruction model, and train the model on Infinigen Indoor dataset for  
 794 10 epochs, with a batch size of 256, and a learning rate of  $10^{-4}$ . It is trained on 64 NVIDIA A100  
 795 GPUs for about 2 days. The checkpoint with the lowest CD loss is selected.

796 To further enhance the robustness and generalization ability of the object-to-code inference model,  
 797 we apply data augmentation techniques. Specifically, we perform random rotation and scaling on the  
 798 objects. Additionally, during training, we randomly sample the number of points in each point cloud  
 799 within the range of 4096 to 16384, and add Gaussian noise to further perturb the input. MeshLLM is  
 800 trained and evaluated on a unified dataset that aggregates all object categories.

### 801 A.4 Complete experiment result of Shape Reconstruction

802 For **MeshLLM**, during inference, each object is represented by a point cloud containing 16,384 points.  
 803 Given the input point cloud, the object-to-code inference model is able to predict the corresponding  
 804 Blender Python script code. The resulting code is then executed to generate a corresponding mesh.  
 805 We uniformly sample 100,000 points from the generated mesh and compute the Chamfer Distance  
 806 (CD) to the input point cloud using the  $L_2$  norm.

807 Given two point sets  $P$  and  $Q$ , each of size 100,000, the  $L_2$  Chamfer Distance is defined as:

$$\text{CD}(P, Q) = \frac{1}{|P|} \sum_{x \in P} \min_{y \in Q} \|x - y\|_2^2 + \frac{1}{|Q|} \sum_{y \in Q} \min_{x \in P} \|y - x\|_2^2.$$

808 To evaluate IoU, we voxelize both the ground-truth mesh and the predicted mesh into grids of  
 809 resolution  $32^3$ , and compute the voxel-based Intersection-over-Union (IoU) as:

$$\text{IoU} = \frac{|\mathcal{V}_{\text{pred}} \cap \mathcal{V}_{\text{gt}}|}{|\mathcal{V}_{\text{pred}} \cup \mathcal{V}_{\text{gt}}|},$$

810 where  $\mathcal{V}_{\text{pred}}$  and  $\mathcal{V}_{\text{gt}}$  denote the sets of occupied voxels in the predicted and ground-truth voxel grids,  
 811 respectively.

812 For **baseline methods**, which take voxel grids as input and output voxel grids, we first voxelize the  
 813 ground-truth mesh into a  $32^3$  grid and feed it into the baseline models. The predicted voxel grid is  
 814 then compared to the input voxelized ground truth to compute IoU. Additionally, we extract a mesh  
 815 from the predicted voxel grid using the Marching Cubes algorithm and uniformly sample 100,000  
 816 points from the resulting mesh surface. These sampled points, along with the ground-truth point  
 817 cloud, are then both uniformly scaled to fit within the  $[-1, 1]^3$  volume. Finally, the Chamfer Distance  
 818 is computed between the two normalized point clouds using the  $L_2$  norm.

819 It’s noticed that for each object category, we independently train the baseline models, according to  
 820 their official code, resulting in category-specific checkpoints. These models are then evaluated on the  
 821 corresponding test sets for each category.

822 The quantitative comparison of reconstruction metrics between MeshLLM and baseline methods  
 823 across all object categories is summarized in Table 2 and Table 3. Some additional examples of object  
 824 reconstruction results and their complete code can be referred to Figure 24, 25, 26.

825 In addition to evaluating our object-to-code inference model, we also perform a quantitative assess-  
 826 ment of our part-to-code inference model. Specifically, for each category described in Section 3.1,  
 827 we construct a test set consisting of 10,000 samples. We evaluate the model’s performance using  
 828 the CD and voxel IoU metrics on these test sets. The results, shown in Table 4, demonstrate strong  
 829 performance across all categories, with low CD values and high IoU scores, indicating that our  
 830 part-to-code inference model is highly effective in generating accurate code representations for  
 831 individual parts.

Table 2: Comparison of reconstruction metrics across all categories. Chamfer Distance (CD) and IoU is shown in percentage (%).

Category	L2 CD( $\times 10^{-2}$ )			Voxel IoU (%)		
	MeshLLM	PLAD	Shape2prog	MeshLLM	PLAD	Shape2prog
ArmChair	0.06	5.54	8.11	92.57	47.17	53.05
BarChair	0.13	2.04	2.53	86.22	38.24	51.37
Bathtub	0.11	1.43	2.22	74.79	53.12	42.86
BeverageFridge	0.22	1.87	3.66	86.29	46.98	54.10
Bottle	0.01	1.44	4.45	88.54	71.18	41.96
Bowl	0.02	1.52	7.09	90.70	50.50	33.32
CeilingClassicLamp	0.03	2.47	4.31	96.16	48.85	45.60
CeilingLight	0.04	1.42	0.71	55.96	34.03	65.88
CellShelf	0.02	1.31	6.93	92.75	56.41	23.87
Lamp	0.01	6.00	32.05	81.53	57.26	18.67
Chair	0.08	2.45	5.19	77.68	36.96	27.83
Chopsticks	0.14	2.53	21.20	75.11	49.42	9.87
Cup	0.06	1.91	8.85	85.21	53.70	27.22
DeskLamp	0.04	1.92	6.57	75.26	59.70	30.35
Dishwasher	0.12	4.71	4.01	85.77	32.24	53.19
FloorLamp	0.01	1.39	19.58	81.00	66.28	15.58
Fork	0.72	0.39	4.29	52.76	72.44	18.03
Hardware	0.02	1.08	4.58	85.52	63.66	37.80
Jar	0.03	0.74	0.82	78.43	66.42	45.51
LargeShelf	0.02	0.66	3.47	81.58	65.32	20.79
Lid	0.07	1.40	3.36	69.01	60.04	51.50
LiteDoor	0.02	8.42	7.29	92.35	60.17	20.41
LouverDoor	0.05	8.15	5.41	88.03	63.76	22.92
Microwave	0.08	3.72	3.86	89.87	50.11	40.58
OfficeChair	0.05	1.68	1.86	73.01	39.29	44.91
PanelDoor	0.02	7.87	3.74	92.92	60.26	24.91
Plate	0.03	0.74	1.67	75.00	61.99	55.14
SidetableDesk	0.02	1.79	6.61	87.98	76.09	48.44
SimpleBookcase	0.03	0.93	4.06	85.68	82.93	30.72
SimpleDesk	0.01	0.60	15.36	82.16	86.79	41.98
Sofa	0.06	2.73	3.05	90.84	44.98	54.88
Spoon	0.03	0.48	2.79	73.80	73.36	19.29
TableCocktail	0.02	3.48	4.89	85.46	41.44	32.07
TableDining	0.03	4.06	1.77	82.34	43.19	62.17
Toilet	0.03	2.53	5.73	86.85	39.64	51.17
TriangleShelf	0.02	1.87	8.11	84.04	44.75	30.15
TV	0.05	1.42	1.83	82.42	52.06	40.71
TVStand	0.02	0.89	4.04	92.84	55.07	28.80
Window	0.07	1.71	2.38	83.29	50.21	53.95
Wineglass	0.02	0.97	6.92	86.40	60.16	29.58
All (Avg.)	<b>0.07</b>	<b>2.83</b>	<b>4.50</b>	<b>83.87</b>	<b>48.42</b>	<b>42.38</b>

#### A.4.1 Ablation Study

We conduct three ablation studies to evaluate the impact of key design choices in our framework.

**Triplane Resolution and the Number of Learnable Tokens.** The first ablation investigates the effect of varying the resolution of the triplane representation and the number of learnable tokens. As shown in Table 5, we observe that increasing both the triplane resolution and the number of learnable tokens consistently improves the performance of the object-to-code inference model. This suggests that a finer-grained spatial encoding and a richer set of token representations enable the model to better capture the underlying 3D structure of objects.

Table 3: Comparison of standard deviation of reconstruction metrics across all categories.

Category	CD			IoU		
	MeshLLM	PLAD	Shape2prog	MeshLLM	PLAD	Shape2prog
ArmChair	$1.67 \times 10^{-3}$	$2.28 \times 10^{-2}$	$2.40 \times 10^{-2}$	$5.35 \times 10^{-2}$	$5.22 \times 10^{-2}$	$6.66 \times 10^{-2}$
BarChair	$2.97 \times 10^{-2}$	$9.70 \times 10^{-3}$	$1.21 \times 10^{-2}$	$9.01 \times 10^{-2}$	$6.72 \times 10^{-2}$	$8.87 \times 10^{-2}$
BathTub	$6.09 \times 10^{-4}$	$1.44 \times 10^{-2}$	$5.32 \times 10^{-3}$	$1.27 \times 10^{-1}$	$9.58 \times 10^{-2}$	$6.99 \times 10^{-2}$
BeverageFridge	$3.34 \times 10^{-3}$	$1.07 \times 10^{-2}$	$8.38 \times 10^{-3}$	$1.07 \times 10^{-1}$	$6.69 \times 10^{-2}$	$5.85 \times 10^{-2}$
Bottle	$6.21 \times 10^{-5}$	$9.71 \times 10^{-3}$	$3.62 \times 10^{-2}$	$1.15 \times 10^{-1}$	$6.32 \times 10^{-2}$	$6.76 \times 10^{-2}$
Bowl	$5.30 \times 10^{-5}$	$4.56 \times 10^{-3}$	$9.04 \times 10^{-3}$	$7.43 \times 10^{-2}$	$5.64 \times 10^{-2}$	$2.97 \times 10^{-2}$
CeilingClassicLamp	$6.00 \times 10^{-7}$	$7.87 \times 10^{-5}$	$2.93 \times 10^{-3}$	$2.95 \times 10^{-5}$	$2.54 \times 10^{-5}$	$1.67 \times 10^{-2}$
CeilingLight	$1.10 \times 10^{-6}$	$1.52 \times 10^{-3}$	$1.50 \times 10^{-3}$	$3.11 \times 10^{-2}$	$3.73 \times 10^{-2}$	$1.45 \times 10^{-2}$
CellShelf	$9.51 \times 10^{-5}$	$6.83 \times 10^{-3}$	$3.00 \times 10^{-2}$	$1.04 \times 10^{-1}$	$8.21 \times 10^{-2}$	$7.61 \times 10^{-2}$
Lamp	$3.75 \times 10^{-4}$	$1.76 \times 10^{-1}$	$3.64 \times 10^{-1}$	$1.79 \times 10^{-1}$	$5.86 \times 10^{-2}$	$1.09 \times 10^{-1}$
Chair	$1.02 \times 10^{-3}$	$7.84 \times 10^{-3}$	$8.32 \times 10^{-2}$	$9.60 \times 10^{-2}$	$6.95 \times 10^{-2}$	$8.15 \times 10^{-2}$
Chopsticks	$1.17 \times 10^{-2}$	$4.28 \times 10^{-2}$	$1.87 \times 10^{-1}$	$2.11 \times 10^{-1}$	$1.09 \times 10^{-1}$	$9.08 \times 10^{-2}$
Cup	$9.49 \times 10^{-4}$	$1.11 \times 10^{-2}$	$5.58 \times 10^{-2}$	$9.86 \times 10^{-2}$	$7.18 \times 10^{-2}$	$9.88 \times 10^{-2}$
DeskLamp	$1.48 \times 10^{-3}$	$5.19 \times 10^{-3}$	$2.52 \times 10^{-2}$	$1.42 \times 10^{-1}$	$5.68 \times 10^{-2}$	$6.47 \times 10^{-2}$
Dishwasher	$4.60 \times 10^{-3}$	$2.89 \times 10^{-2}$	$1.22 \times 10^{-2}$	$1.29 \times 10^{-1}$	$1.33 \times 10^{-1}$	$5.96 \times 10^{-2}$
FloorLamp	$8.07 \times 10^{-5}$	$4.91 \times 10^{-2}$	$2.94 \times 10^{-1}$	$1.87 \times 10^{-1}$	$6.05 \times 10^{-2}$	$1.06 \times 10^{-1}$
Fork	$4.47 \times 10^{-2}$	$1.09 \times 10^{-3}$	$3.92 \times 10^{-2}$	$1.85 \times 10^{-1}$	$7.77 \times 10^{-2}$	$1.13 \times 10^{-1}$
Hardware	$1.48 \times 10^{-4}$	$8.02 \times 10^{-3}$	$5.89 \times 10^{-2}$	$1.32 \times 10^{-1}$	$1.40 \times 10^{-1}$	$1.12 \times 10^{-1}$
Jar	$1.54 \times 10^{-4}$	$2.19 \times 10^{-3}$	$5.01 \times 10^{-3}$	$1.48 \times 10^{-1}$	$5.78 \times 10^{-2}$	$6.88 \times 10^{-2}$
LargeShelf	$4.99 \times 10^{-5}$	$2.38 \times 10^{-3}$	$2.41 \times 10^{-2}$	$1.52 \times 10^{-1}$	$9.99 \times 10^{-2}$	$4.89 \times 10^{-2}$
Lid	$5.19 \times 10^{-4}$	$1.14 \times 10^{-2}$	$1.77 \times 10^{-2}$	$1.46 \times 10^{-1}$	$1.15 \times 10^{-1}$	$1.17 \times 10^{-1}$
LiteDoor	$4.26 \times 10^{-3}$	$4.93 \times 10^{-2}$	$8.22 \times 10^{-2}$	$1.68 \times 10^{-1}$	$1.06 \times 10^{-1}$	$1.15 \times 10^{-1}$
LouverDoor	$1.08 \times 10^{-3}$	$4.45 \times 10^{-2}$	$8.01 \times 10^{-2}$	$1.83 \times 10^{-1}$	$1.08 \times 10^{-1}$	$1.27 \times 10^{-1}$
Microwave	$3.91 \times 10^{-3}$	$2.43 \times 10^{-2}$	$1.91 \times 10^{-2}$	$7.61 \times 10^{-2}$	$1.34 \times 10^{-1}$	$5.08 \times 10^{-2}$
OfficeChair	$1.10 \times 10^{-3}$	$5.80 \times 10^{-3}$	$2.30 \times 10^{-2}$	$9.07 \times 10^{-2}$	$5.49 \times 10^{-2}$	$6.53 \times 10^{-2}$
PanelDoor	$3.13 \times 10^{-3}$	$5.45 \times 10^{-2}$	$3.01 \times 10^{-2}$	$1.63 \times 10^{-1}$	$1.00 \times 10^{-1}$	$2.86 \times 10^{-1}$
Plate	$1.31 \times 10^{-4}$	$3.94 \times 10^{-3}$	$8.54 \times 10^{-3}$	$1.74 \times 10^{-1}$	$9.83 \times 10^{-2}$	$1.26 \times 10^{-1}$
SidetableDesk	$1.56 \times 10^{-4}$	$5.01 \times 10^{-3}$	$7.03 \times 10^{-2}$	$1.33 \times 10^{-1}$	$1.37 \times 10^{-1}$	$1.68 \times 10^{-1}$
SimpleBookcase	$6.34 \times 10^{-5}$	$7.55 \times 10^{-3}$	$1.82 \times 10^{-2}$	$1.19 \times 10^{-1}$	$1.12 \times 10^{-1}$	$6.01 \times 10^{-2}$
SimpleDesk	$8.67 \times 10^{-5}$	$3.63 \times 10^{-2}$	$1.23 \times 10^{-1}$	$1.91 \times 10^{-1}$	$1.57 \times 10^{-1}$	$1.23 \times 10^{-1}$
Sofa	$2.81 \times 10^{-3}$	$1.62 \times 10^{-2}$	$1.36 \times 10^{-2}$	$8.71 \times 10^{-2}$	$8.19 \times 10^{-2}$	$6.98 \times 10^{-2}$
Spoon	$5.55 \times 10^{-4}$	$1.75 \times 10^{-3}$	$3.72 \times 10^{-2}$	$1.78 \times 10^{-1}$	$9.63 \times 10^{-2}$	$9.17 \times 10^{-2}$
TableCocktail	$1.95 \times 10^{-4}$	$3.34 \times 10^{-2}$	$6.12 \times 10^{-2}$	$1.17 \times 10^{-1}$	$1.41 \times 10^{-1}$	$1.55 \times 10^{-1}$
TableDining	$3.83 \times 10^{-3}$	$3.34 \times 10^{-2}$	$1.55 \times 10^{-2}$	$1.71 \times 10^{-1}$	$1.70 \times 10^{-1}$	$1.64 \times 10^{-1}$
Toilet	$9.62 \times 10^{-5}$	$9.11 \times 10^{-3}$	$2.41 \times 10^{-2}$	$4.82 \times 10^{-2}$	$5.82 \times 10^{-2}$	$3.04 \times 10^{-2}$
TriangleShelf	$4.67 \times 10^{-5}$	$4.69 \times 10^{-3}$	$2.13 \times 10^{-2}$	$1.15 \times 10^{-1}$	$5.40 \times 10^{-2}$	$8.07 \times 10^{-2}$
TV	$6.61 \times 10^{-4}$	$1.28 \times 10^{-2}$	$1.22 \times 10^{-2}$	$1.85 \times 10^{-1}$	$1.48 \times 10^{-1}$	$1.07 \times 10^{-1}$
TVStand	$8.12 \times 10^{-5}$	$3.39 \times 10^{-3}$	$1.88 \times 10^{-2}$	$1.23 \times 10^{-1}$	$8.85 \times 10^{-2}$	$6.46 \times 10^{-2}$
Window	$3.90 \times 10^{-3}$	$3.11 \times 10^{-2}$	$4.49 \times 10^{-2}$	$1.89 \times 10^{-1}$	$1.62 \times 10^{-1}$	$1.40 \times 10^{-1}$
Wineglass	$1.86 \times 10^{-4}$	$2.91 \times 10^{-3}$	$3.79 \times 10^{-2}$	$1.04 \times 10^{-1}$	$5.53 \times 10^{-2}$	$7.33 \times 10^{-2}$
All (Std.)	$8.66 \times 10^{-3}$	$3.22 \times 10^{-2}$	$6.57 \times 10^{-2}$	$1.43 \times 10^{-1}$	$1.40 \times 10^{-1}$	$1.63 \times 10^{-1}$

Table 4: Quantitative evaluation of the *part-to-code* inference model across different part categories. CD is reported in  $10^{-2}$ , and IoU is reported in percentage.

Category	CD ( $\times 10^{-2}$ )	IoU (%)
Primitive	0.18	94.81
Boolean	0.03	96.13
Array	0.70	78.90
Bridge Loop	0.14	89.16
Translation	0.17	83.45

840 **Initialization from Part-to-Code Checkpoint.** The second ablation study evaluates whether  
841 initializing the object-to-code model with the pre-trained checkpoint of the part-to-code inference  
842 model yields performance improvements. Table 6 demonstrates that such initialization leads to  
843 noticeably better results. This improvement may be attributed to the part-to-code model’s ability to  
844 learn robust 3D geometric representations and syntactic grammar structures from the diverse part-level  
845 dataset. These learned features likely provide transferable knowledge that facilitates generalization  
846 during the object-level inference process, thereby improving the effectiveness of the model.

Table 5: Ablation study on triplane resolution and the number of learnable tokens in MeshLLM. We report L2 Chamfer Distance ( $\times 10^{-4}$ ) and IoU (%).

Triplane Resolution	Token Number	L2 CD ( $\times 10^{-4}$ )	IoU (%)
128	128	<b>6.86</b>	<b>83.87</b>
128	64	7.82	82.61
128	32	11.55	82.09
64	128	7.47	82.31

Table 6: Ablation study on whether to initialize object-to-code model from the part-to-code checkpoint.

Initialization Strategy	L2 CD ( $\times 10^{-4}$ )	IoU (%)
From Scratch	20.25	80.28
From Part-to-Code Checkpoint	<b>6.86</b>	<b>83.87</b>

**Using Learnable Tokens as Queries.** The third ablation study explores the role of learnable tokens when used as queries for the LLM. In our default setup, learnable tokens are used as input queries to the LLM, whereas in the alternative setup, the triplane-encoded features are passed through an MLP and directly fed into the LLM. As reported in Table 7, the learnable-token-as-query strategy achieves superior performance. We hypothesize that this advantage arises for two reasons: (1) the learnable tokens are capable of aggregating global information across the entire input, unlike the direct feature approach where each patch predominantly captures localized information, and (2) the learnable tokens can adaptively organize the input representation in a layout that is more aligned with the LLM’s internal understanding and processing patterns.

Table 7: Ablation study on whether to use learnable tokens as queries in the transformer.

Query Type	L2 CD ( $\times 10^{-4}$ )	IoU (%)
MLP Projection Only	20.09	79.83
Learnable Tokens (Ours)	<b>6.86</b>	<b>83.87</b>

## A.5 Complete experiment result of Shape Editing

We additionally present two examples of shape editing along with their complete code implementations. In Figure 27, we modify the thickness of the chair legs and armrests by adjusting the `scale` parameter. In Figure 28, we change the mesh resolution of a plate by modifying the `resolution` parameter.

## A.6 Complete experiment result of Shape Understanding

When presented with a 3D point cloud of an object as input, MeshLLM can infer the corresponding code for the object. Upon execution of this code in Blender, the geometry of the object can be obtained. Notably, the comments within the code encompass a variety of semantically rich cues, such as the object’s identity and the specifics of each component. The primary aim of this experiment is to highlight that our model can assist existing large language models, like GPT - 4, in understanding the structure of 3D objects. We provide the inferred code to GPT - 4 and then inquire about the geometry or structure of the object, as showed in Figure 19, Figure 20 and Figure 21. GPT - 4 is able to generate relevant responses based on the code inferred by our model. This demonstrates that our model possesses capabilities in understanding the geometry and structure of 3D objects and can aid large - scale models such as GPT in addressing such questions. However, our model does have limitations. Currently, the code inferred by our model solely contains geometric information of the object and does not include color information. As a result, it is unable to answer questions pertaining to color.



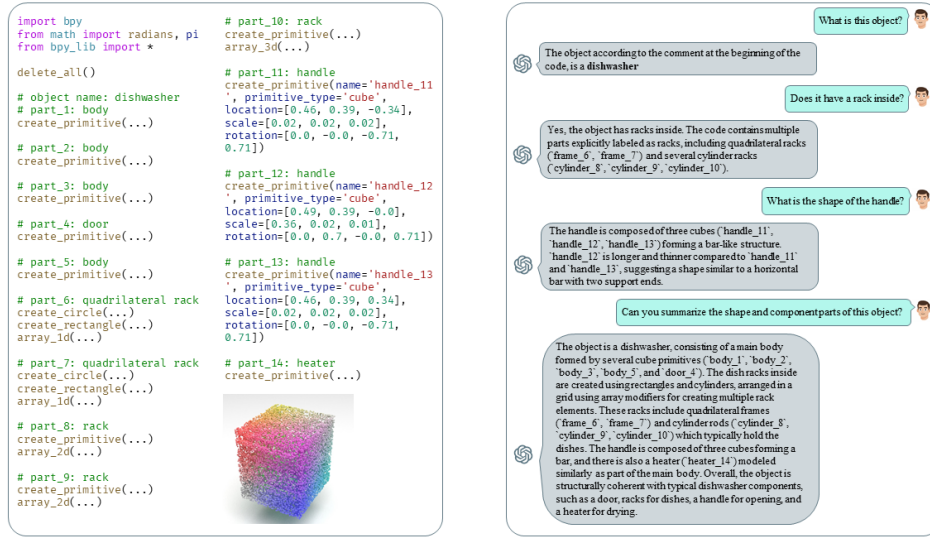


Figure 19: Experiments on how GPT-4o can understand shape through given dishwasher code.

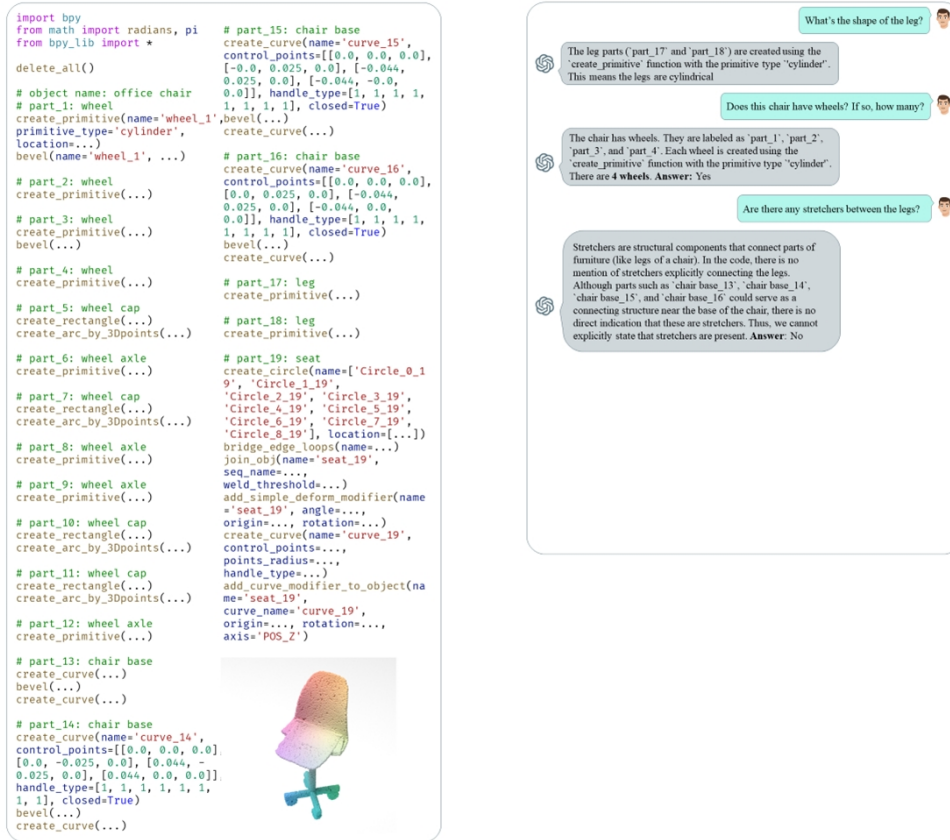


Figure 20: Experiments on how GPT-4o can understand shape through given office chair code.

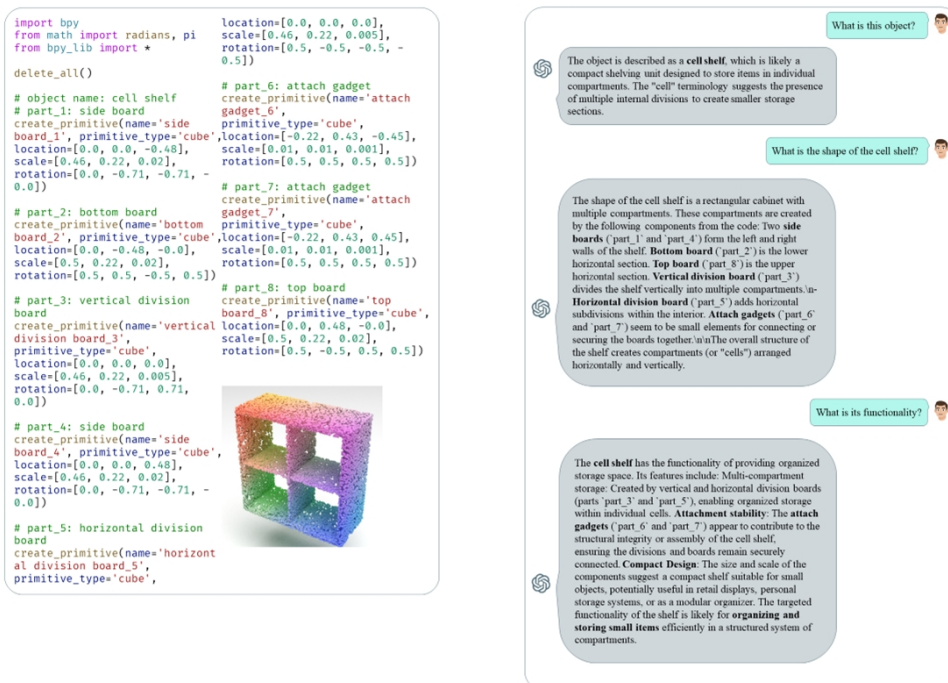


Figure 21: Experiments on how GPT-4o can understand shape through given cell shelf code.

```

"""
Create primitive object
"""
def create_primitive(name, primitive_type="cube", location=None, scale=None, rotation=None, rotation_mode='QUATERNION', apply=False,
x_subdivisions=None, y_subdivisions=None, use_minimum_face=USE_MINIMUM_FACE, average_edge_length=AVERAGE_EDGE_LENGTH, resolution=None):

    if primitive_type=="uv_sphere":
        if not use_minimum_face:
            if average_edge_length !=None:
                res=int(2*pi//average_edge_length)
                segments, ring_count=res, res
            else:
                segments, ring_count=resolution[0], resolution[1]
            getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")(segments=segments,ring_count=ring_count)
            primitive = bpy.context.object
            primitive.name = name
        else:
            getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")()
            primitive = bpy.context.object
            primitive.name = name

    if primitive_type in ["cylinder","cone"]:
        if not use_minimum_face:
            if average_edge_length !=None:
                res_1=int(2*pi//average_edge_length)
                vertices=res_1
                res_2=int(2//average_edge_length)
            else:
                vertices,res_2=resolution[0], resolution[1]
            getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")(vertices=vertices)
            primitive = bpy.context.object
            primitive.name = name
            if primitive_type=="cylinder":
                primitive=subdivide_primitive(name,[res_2,['Z']])
            else:
                primitive=split_cone_z(name,res_2)
        else:
            getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")()
            primitive = bpy.context.object
            primitive.name = name

    if primitive_type=="torus":
        if not use_minimum_face:
            if average_edge_length !=None:
                res_1=int(2*pi//average_edge_length)
                res_2=int(0.5*pi//average_edge_length)
            else:
                res_1, res_2=resolution[0], resolution[1]
            getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")(major_segments=res_1,minor_segments=res_2)
            primitive = bpy.context.object
            primitive.name = name
        else:
            getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")()
            primitive = bpy.context.object
            primitive.name = name

    if primitive_type=="cube":
        if not use_minimum_face:
            if average_edge_length !=None:
                res=int(2//average_edge_length)
                if res>1:
                    resolution=[res,res,res]
            else:
                pass
            getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")()
            primitive = bpy.context.object
            primitive.name = name
            primitive=subdivide_primitive(name,resolution,['X','Y','Z'])
        else:
            getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")()
            primitive = bpy.context.object
            primitive.name = name

    if primitive_type=="grid":
        getattr(bpy.ops.mesh, f"primitive_{primitive_type}_add")(x_subdivisions=x_subdivisions,y_subdivisions=y_subdivisions)
        primitive = bpy.context.object
        primitive.name = name

    if location:
        primitive.location = location
    if scale:
        primitive.scale = scale
    if rotation:
        if rotation_mode=="XYZ":
            primitive.rotation_euler = [angle * pi for angle in rotation]
        elif rotation_mode=='QUATERNION':
            primitive.rotation_mode = 'QUATERNION'
            primitive.rotation_quaternion = rotation
        elif rotation_mode=='MATRIX':
            mat = np.eye(4)
            rotation = np.array(rotation).reshape([3,3])
            mat[:3,:3] = rotation
            bpy.context.view_layer.update()
            world_matrix = torch.tensor(bpy.data.objects[name].matrix_world)
            scale_now = world_matrix.norm(dim=0)[:3]
            scale_matrix = torch.eye(4)
            scale_matrix[0,0],scale_matrix[1,1],scale_matrix[2,2] = scale_now[0],scale_now[1],scale_now[2]
            scale_matrix_inv = scale_matrix.clone()
            for i in range(3):
                if scale_matrix_inv[i,i]>1e-10:
                    scale_matrix_inv[i,i]=1.0 / scale_matrix_inv[i,i]
            mat = scale_matrix_inv@torch.tensor(mat, dtype=torch.float32)@scale_matrix
            mat = mathutils.Matrix(np.array(mat))
            bpy.data.objects[name].matrix_world = bpy.data.objects[name].matrix_world@mat

    if apply:
        bpy.ops.object.transform_apply(location=True, rotation=True, scale=True)

    return primitive

```

Figure 22: Implementation of the function for creating primitives

```

"""
Creates a translational object of a line trajectory
"""
def create_curve(name, profile_name=None, control_points=[], points_radius=[], handle_type=[], closed=False, center="POINT", thickness=None,
fill_caps="none", flip_normals=False, bevel_width=None, bevel_segments=8, use_minimum_face=USE_MINIMUM_FACE, average_edge_length=AVERAGE_EDGE_LENGTH, resolution=24,
volum_origin=True):
    if isinstance(name, str):
        type_dict={0:"AUTO", 1:"VECTOR", 2:"ALIGNED", 3:"FREE"}

        control_points = np.array(control_points).tolist()
        control_points_tmp = copy.deepcopy(control_points)
        num_handle_co = handle_type.count(3)
        num_control_points = len(control_points) - num_handle_co

        curveData = bpy.data.curves.new(name, type='CURVE')
        curveData.dimensions = '3D'

        bezierSpline = curveData.splines.new('BEZIER')
        bezierSpline.bezier_points.add(num_control_points - 1)
        bezierSpline.use_cyclic_u = closed

        for i in range(num_control_points):
            bezier_point = bezierSpline.bezier_points[i]
            bezier_point.handle_left_type = type_dict[handle_type[2*i]]
            if type_dict[handle_type[2*i]]=="FREE":
                bezier_point.handle_left = control_points.pop(0)
            bezier_point.co = control_points.pop(0)
            bezier_point.handle_right_type = type_dict[handle_type[2*i+1]]
            if type_dict[handle_type[2*i+1]]=="FREE":
                bezier_point.handle_right = control_points.pop(0)
            bezier_point.radius = points_radius[i] if len(points_radius)!=0 else 1.0

        assert len(control_points)==0, "cannot create curve"
        if use_minimum_face:
            use_resolution = 12
        elif not average_edge_length is None:
            for i in range(len(bezierSpline.bezier_points) - 1):
                p1 = bezierSpline.bezier_points[i].co
                p2 = bezierSpline.bezier_points[i + 1].co
                total_length += (p2 - p1).length
            use_resolution = total_length/average_edge_length

        if resolution:
            use_resolution = resolution
        curveData.resolution_u = use_resolution

        curveOB = bpy.data.objects.new(name, curveData)

        if profile_name != None:
            curveData.bevel_mode = "OBJECT"
            curveData.splines[0].use_smooth = False
            scn = bpy.context.scene.collection
            scn.objects.link(curveOB)

            if bevel_width!=None:
                bevel(name=name, width=bevel_width, segments=bevel_segments)
                curveData = bpy.data.objects[name].data
                curveData.bevel_mode = 'OBJECT'

            curveData.bevel_object = bpy.data.objects[profile_name]
            if fill_caps=="both":
                curveData.use_fill_caps = True
            else:
                curveData.use_fill_caps = False

            if use_minimum_face:
                use_resolution = 24
            elif not average_edge_length is None:
                for i in range(len(bezierSpline.bezier_points) - 1):
                    p1 = bezierSpline.bezier_points[i].co
                    p2 = bezierSpline.bezier_points[i + 1].co
                    total_length += (p2 - p1).length
                use_resolution = total_length/average_edge_length

            if resolution:
                use_resolution = resolution

            curveData.resolution_u = use_resolution

            bpy.context.view_layer.objects.active = bpy.data.objects[name]
            bpy.ops.object.mode_set(mode = 'OBJECT')
            bpy.data.objects[name].select_set(True)
            bpy.ops.object.convert(target='MESH')
            bpy.data.objects.remove(bpy.data.objects[profile_name], do_unlink=True)
            if volum_origin:
                bpy.ops.object.origin_set(type='ORIGIN_CENTER_OF_VOLUME', center='MEDIAN')

            if fill_caps in ["start", "end"]:
                make_caps(name, fill_caps)

            if flip_normals:
                recalculate_normals(name, inside=True)
            else:
                recalculate_normals(name, inside=False)

            if thickness>1e-10:
                solidify(name, thickness)

            weld(name, 1e-5)

            return curveOB
        else:
            scn = bpy.context.scene.collection
            scn.objects.link(curveOB)
            if center=="MEDIAN":
                bpy.data.objects[name].select_set(True)
                bpy.ops.object.origin_set(type='ORIGIN_GEOMETRY', center='MEDIAN')
            points=np.array(control_points_tmp)
            return {"name":name, "points":points, "handle_type":handle_type, "closed":closed, "center":center}

    elif profile_name==None:
        if isinstance(profile_name, str):
            profile_name = [profile_name]*len(name)
        if len(points_radius) != 0 and (isinstance(points_radius[0], float) or isinstance(points_radius[0], int)):
            points_radius = [points_radius]*len(name)
        elif len(points_radius) == 0:
            points_radius = [[]] * len(name)
        if isinstance(handle_type[0], int):
            handle_type = [handle_type]*len(name)
        if isinstance(closed, bool):
            closed = [closed]*len(name)
        if isinstance(center, str):
            center = [center]*len(name)
        for i in range(len(name)):
            create_curve(name=name[i], control_points=control_points[i], points_radius=points_radius[i], handle_type=handle_type[i], closed=closed[i],
            center=center[i], thickness=thickness, fill_caps=fill_caps, flip_normals=flip_normals, resolution=resolution)
        points = np.array(copy.deepcopy(control_points))
        return {"name":name, "points":points, "handle_type":handle_type, "closed":closed, "center":center}

```

Figure 23: Implementation of the function for creating curves





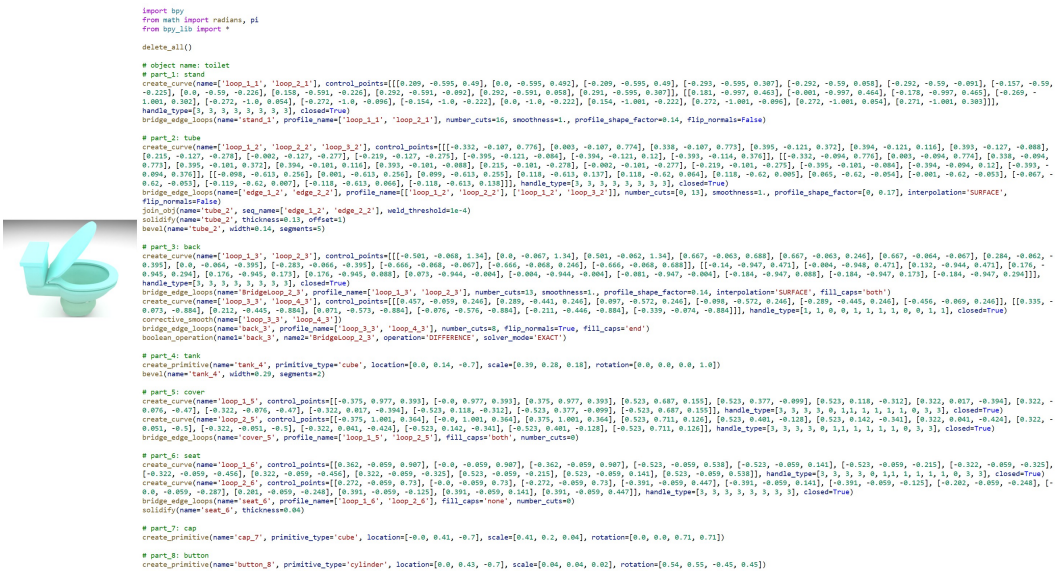


Figure 26: An example of toilet. The input is a point cloud of a toilet, and the figure shows the code inferred by the object-to-code inference model, as well as the resulting mesh generated by executing the inferred code.



```
import bpy
from math import radians, pi
from bpy_lib import *

delete_all()

# object name: chair
# part.1: leg
create_primitive(name='leg_1', primitive_type='cube', location=[-0.44, -0.46, 0.37], scale=[0.53, 0.05, 0.05], rotation=[0.5, 0.51, -0.51, -0.49])
bevel(name='leg_1', width=0.12, segments=8)

# part.2: leg
create_primitive(name='leg_2', primitive_type='cube', location=[-0.31, -0.46, -0.46], scale=[0.53, 0.05, 0.05], rotation=[0.51, -0.5, -0.49, 0.51])

# part.3: leg
create_primitive(name='leg_3', primitive_type='cube', location=[0.31, -0.46, -0.46], scale=[0.53, 0.05, 0.05], rotation=[0.51, -0.5, -0.49, 0.51])

# part.4: leg
create_primitive(name='leg_4', primitive_type='cube', location=[0.44, -0.46, 0.37], scale=[0.53, 0.05, 0.05], rotation=[0.5, 0.51, -0.51, -0.49])

# part.5: leg decoration
create_primitive(name='leg decoration_5', primitive_type='cube', location=[-0.37, -0.35, -0.05], scale=[0.42, 0.05, 0.05], rotation=[0.76, 0.01, 0.65, -0.01])
bevel(name='leg decoration_5', width=0.13, segments=1)

# part.6: leg decoration
create_primitive(name='leg decoration_6', primitive_type='cube', location=[0.37, -0.35, -0.05], scale=[0.42, 0.05, 0.05], rotation=[0.65, -0.01, 0.76, 0.01])
bevel(name='leg decoration_6', width=0.15, segments=6)

# part.7: seat
create_curve(name='seat_7', control_points=[[0.0, 0.03, -0.51], [-0.35, 0.03, -0.51], [-0.47, 0.05, 0.21], [-0.49, 0.03, 0.41], [0.0, 0.03, 0.5], [0.49, 0.03, 0.41], [0.47, 0.05, 0.21], [0.35, 0.03, -0.51], [0.0, 0.03, -0.51]], handle_type=[0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], fill_grid(name='seat_7', thickness=0.1042)
bevel(name='seat_7', width=0.05, segments=1)

# part.8: arm
create_circle(name='circle_8', radius=0.08, center='MEDIAN')
create_curve(name='arm_8', profile_name='circle_8', control_points=[[-0.33, 0.64, -0.46], [-0.39, 0.64, -0.2], [-0.46, 1.02, -0.06], [-0.46, 0.64, 0.21], [-0.46, 0.1, 0.31]], points_radius=[1.0, 1.0, 1.0], handle_type=[0, 3, 3, 1, 1, 0], thickness=0.001, fill_caps='both')

# part.9: back
create_primitive(name='back_9', primitive_type='cube', location=[-0.31, 0.54, -0.46], scale=[0.45, 0.04, 0.04], rotation=[0.5, 0.51, -0.51, -0.49])

# part.10: back
create_primitive(name='back_10', primitive_type='cube', location=[0.31, 0.54, -0.46], scale=[0.45, 0.04, 0.04], rotation=[0.5, 0.51, 0.51, 0.49])

# part.11: arm
create_circle(name='circle_11', radius=0.08, center='MEDIAN')
create_curve(name='arm_11', profile_name='circle_11', control_points=[[-0.33, 0.64, -0.46], [-0.39, 0.64, -0.21], [-0.46, 1.02, -0.06], [-0.46, 0.64, 0.21], [-0.46, 0.09, 0.32]], points_radius=[1.0, 1.0, 1.0], handle_type=[0, 3, 3, 1, 1, 0], thickness=0.001, fill_caps='both')

# part.12: back decoration
create_curve(name='back decoration_12', control_points=[[-0.35, 0.8, -0.51], [-0.35, 0.62, -0.51], [0.01, 0.62, -0.51], [0.35, 0.62, -0.51], [0.35, 0.8, -0.51], [0.35, 0.99, -0.51], [0.01, 0.99, -0.51], [-0.35, 0.99, -0.51], [-0.35, 0.8, -0.51]], handle_type=[0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0], fill_grid(name='back decoration_12', thickness=0.086)
bevel(name='back decoration_12', width=0.04, segments=10)
```



```
import bpy
from math import radians, pi
from bpy_lib import *

delete_all()

# object name: chair
# part.1: leg
create_primitive(name='leg_1', primitive_type='cube', location=[-0.44, -0.46, 0.37], scale=[0.53, 0.08, 0.08], rotation=[0.5, 0.51, -0.51, -0.49])
bevel(name='leg_1', width=0.12, segments=8)

# part.2: leg
create_primitive(name='leg_2', primitive_type='cube', location=[-0.31, -0.46, -0.46], scale=[0.53, 0.08, 0.08], rotation=[0.51, -0.5, -0.49, 0.51])

# part.3: leg
create_primitive(name='leg_3', primitive_type='cube', location=[0.31, -0.46, -0.46], scale=[0.53, 0.08, 0.08], rotation=[0.51, -0.5, -0.49, 0.51])

# part.4: leg
create_primitive(name='leg_4', primitive_type='cube', location=[0.44, -0.46, 0.37], scale=[0.53, 0.08, 0.08], rotation=[0.5, 0.51, -0.51, -0.49])

# part.5: leg decoration
create_primitive(name='leg decoration_5', primitive_type='cube', location=[-0.37, -0.35, -0.05], scale=[0.42, 0.05, 0.05], rotation=[0.76, 0.01, 0.65, -0.01])
bevel(name='leg decoration_5', width=0.13, segments=1)

# part.6: leg decoration
create_primitive(name='leg decoration_6', primitive_type='cube', location=[0.37, -0.35, -0.05], scale=[0.42, 0.05, 0.05], rotation=[0.65, -0.01, 0.76, 0.01])
bevel(name='leg decoration_6', width=0.15, segments=6)

# part.7: seat
create_curve(name='seat_7', control_points=[[0.0, 0.03, -0.51], [-0.35, 0.03, -0.51], [-0.47, 0.05, 0.21], [-0.49, 0.03, 0.41], [0.0, 0.03, 0.5], [0.49, 0.03, 0.41], [0.47, 0.05, 0.21], [0.35, 0.03, -0.51], [0.0, 0.03, -0.51]], handle_type=[0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], fill_grid(name='seat_7', thickness=0.1042)
bevel(name='seat_7', width=0.05, segments=1)

# part.8: arm
create_circle(name='circle_8', radius=0.08, center='MEDIAN')
create_curve(name='arm_8', profile_name='circle_8', control_points=[[-0.33, 0.64, -0.46], [-0.39, 0.64, -0.2], [-0.46, 1.02, -0.06], [-0.46, 0.64, 0.21], [-0.46, 0.1, 0.31]], points_radius=[1.0, 1.0, 1.0], handle_type=[0, 3, 3, 1, 1, 0], thickness=0.001, fill_caps='both')

# part.9: back
create_primitive(name='back_9', primitive_type='cube', location=[-0.31, 0.54, -0.46], scale=[0.45, 0.04, 0.04], rotation=[0.5, 0.51, -0.51, -0.49])

# part.10: back
create_primitive(name='back_10', primitive_type='cube', location=[0.31, 0.54, -0.46], scale=[0.45, 0.04, 0.04], rotation=[0.5, 0.51, 0.51, 0.49])

# part.11: arm
create_circle(name='circle_11', radius=0.08, center='MEDIAN')
create_curve(name='arm_11', profile_name='circle_11', control_points=[[-0.33, 0.64, -0.46], [-0.39, 0.64, -0.21], [-0.46, 1.02, -0.06], [-0.46, 0.64, 0.21], [-0.46, 0.09, 0.32]], points_radius=[1.0, 1.0, 1.0], handle_type=[0, 3, 3, 1, 1, 0], thickness=0.001, fill_caps='both')

# part.12: back decoration
create_curve(name='back decoration_12', control_points=[[-0.35, 0.8, -0.51], [-0.35, 0.62, -0.51], [0.01, 0.62, -0.51], [0.35, 0.62, -0.51], [0.35, 0.8, -0.51], [0.35, 0.99, -0.51], [0.01, 0.99, -0.51], [-0.35, 0.99, -0.51], [-0.35, 0.8, -0.51]], handle_type=[0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0], fill_grid(name='back decoration_12', thickness=0.086)
bevel(name='back decoration_12', width=0.04, segments=10)
```

Figure 27: By modifying the scale parameters of the leg and arm parts, we adjust their thickness. The highlighted sections indicate the changes made.

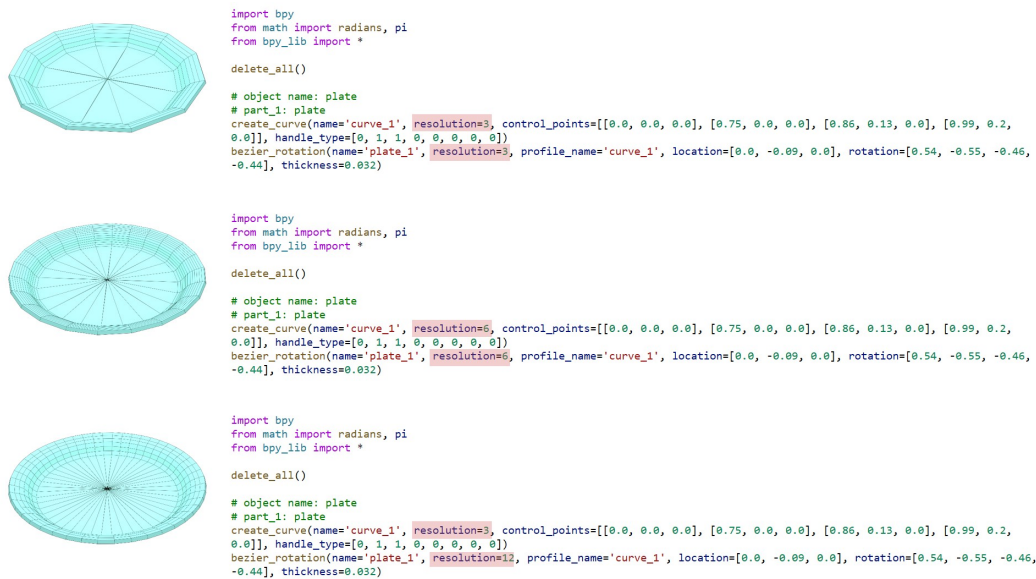


Figure 28: By modifying the resolution parameter, we change its resolution. The highlighted sections indicate the changes made.