Combinatorial 3D Shape Generation via Sequential Assembly

Jungtaek Kim
POSTECH
jtkim@postech.ac.kr

Hyunsoo Chung
POSTECH
hschung2@postech.ac.kr

Jinhwi Lee
POSTECH, POSCO
jinhwi@postech.ac.kr

Minsu Cho
POSTECH
mscho@postech.ac.kr

Jaesik Park
POSTECH
jaesik.park@postech.ac.kr

Abstract

Sequential assembly with geometric primitives has drawn attention in robotics and 3D vision since it yields a practical blueprint to construct a target shape. However, due to its combinatorial property, a greedy method falls short of generating a sequence of volumetric primitives. To alleviate this consequence induced by a huge number of feasible combinations, we propose a combinatorial 3D shape generation framework. The proposed framework reflects an important aspect of human generation processes in real life – we often create a 3D shape by sequentially assembling unit primitives with geometric constraints. To find the desired combination regarding combination evaluations, we adopt Bayesian optimization, which is able to exploit and explore efficiently the feasible regions constrained by the current primitive placements. An evaluation function conveys global structure guidance for an assembly process and stability in terms of gravity and external forces simultaneously. Experimental results demonstrate that our method successfully generates combinatorial 3D shapes and simulates more realistic generation processes. We also introduce a new dataset for combinatorial 3D shape generation. All the codes are available at `https://github.com/POSTECH-CVLab/Combinatorial-3D-Shape-Generation`.

1 Introduction

Constructing a 3D shape via sequential assembly has a huge potential. This generation scheme aims to follow a target shape that can be employed in mimicking a human assembling process and allocating a budget of the number of primitives given. In robotics, self-assembly robots [36, 30, 11, 31], inspired by a phenomenon that self-organize a chaotic pattern to an ordered structure in chemistry [37] and biology [14], are used in creating a target shape under a specific form of geometric constraints such as contacts and local interactions. This line of research demonstrates the impressive results that open new applications of sequential assembly. However, these methods have the limitation that they adopt a heuristic or fixed-strategy approach to construct a target shape.

Generic generative models such as variational auto-encoders and generative adversarial networks have been used where the following 3D representation is assumed; a fixed number of points [1], a deformable mesh [12, 13], or a voxel grid [38, 39]. However, these generation schemes do not reflect an important aspect of human assembling processes in real life – we often create a 3D shape by sequentially assembling primitives into a combinatorial configuration. In this work, we solve a sequential assembling problem with Bayesian optimization-based framework of *combinatorial 3D*

Workshop on machine learning for engineering modeling, simulation and design @ NeurIPS 2020

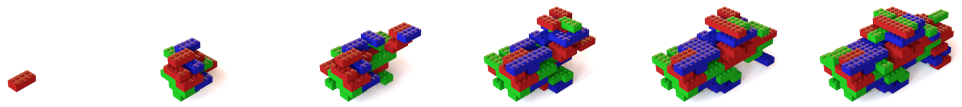


Figure 1: Generated assembling sequence that creates a *car* shape with 118 unit primitives.

shape generation that creates a 3D shape with a set of *geometric primitives*. Our method can generate a sequence of unit primitives without a significant amount of human effort or a brute-force technique.

In practice, the main challenge lies in a combinatorial explosion as the number of primitives increasing. For instance, if we assemble 2×4 LEGO bricks on a free 3D space, the most naïve way is to take all combinations into account and pick the most probable one for the purpose. However, with only six 2×4 LEGO bricks, the number of candidates amounts to 915 million combinations [9].[1] A brute-force approach to combinatorial generation is to find a needle in a haystack due to the prohibitive number of possible combinations. To tackle this challenge, we propose a sequential assembling method that iteratively evaluates the next possible primitive combinations in a sample-efficient manner, by considering global structure guidance for assembling a target shape and stability in terms of gravity and external forces. In addition to the proposed pipeline, we introduce a new combinatorial 3D shape dataset that consists of 14 classes and 406 instances. Due to the nature of the combinatorial shape, the dataset can be readily augmented by manipulating assembling orders. We hope the new dataset opens a new benchmark for 3D shape generation.

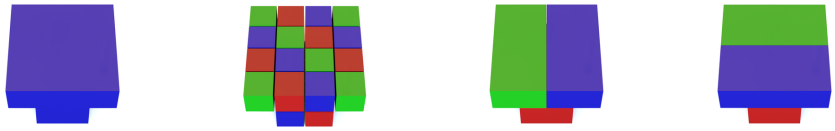
2 Related Work

Self-Assembly Robots This class of robots has been inspired by a self-assembly phenomenon of the nature [37, 14], which is driven by physical interaction between molecules or unit components, or surrounding environments. Wei et al. [36] present a self-configurable and self-assembly modular robot, the actuator of which can enable itself to move and dock to other modules. Romanishin et al. [30] propose a magnetic modular robot, which can move independently with the flywheels installed in the edges of robot. [11] suggests a printed self-folding robot that can be used in various fields such as micro-assembly and space applications. Rubenstein et al. [31] propose a flock of programmable self-assembly robots, which can create a target shape without external intervention.

Topology and Layout Optimization Topology optimization [3], which finds an optimal layout where predefined configurations and constraints are provided, has widely been used in shape design, prototyping, and manufacturing. Eschenauer et al. [10] introduce a method that inserts holes into a component with iterative positioning. Borrvall and Petersson [4] use a topology optimization technique in fluid mechanics to solve applications such as pipe and airfoil designs. Kharmanda et al. [20] suggest a method to find reliable and efficient structures with the reliability index. Brackett et al. [5] utilize topology optimization in additive manufacturing for producing end-use parts. Moreover, to find an optimal layout for 3D objects, Testuz et al. [35] identify a suitable primitive set for a given mesh and applies a greedy method to repair weak connections. Lee et al. [22] propose to optimize a primitive layout using genetic algorithm. Luo et al. [24] consider the physical stability of constructed model, which helps to create realistic and realizable assembly accomplishments.

Generative Models for 3D Objects Achlioptas et al. [1] learn representations from point clouds via autoencoder. Their approach employs either raw point sets or learned representations, to train a generative adversarial network. For the deformable mesh generation, Groueix et al. [13] suggest a method to transform 2D texture map atlases into 3D surface. Gao et al. [12] generate structured deformable meshes. The network is composed of part-level and structural-level variational autoencoder. On the other hand, convolutional deep belief networks [39] and generative adversarial networks [38] are used to generate an occupancy grid. Nash et al. [26] predict mesh vertices and faces using Transformer-based neural networks.

[1] Under the assembling conditions suggested in this paper; there exist 46 combinations for two bricks, 3,566 for three bricks, 405,716 for four bricks, and 59,814,648 for five bricks.



(a) Target shape (b) 1×1-sized primitives (c) 2×4-sized primitives

Figure 2: Results on assembling a target shape with 1×1-sized primitives and 2×4-sized primitives.

Compared to the aforementioned techniques that attempt to find an optimal layout in terms of their own evaluation metrics, our goal is to create a sequence of unit primitives, considering the stability of combination as well as following a desired shape. To the best of our knowledge, this is the first attempt to build a 3D shape via a sequential and combinatorial approach. In particular, our algorithm efficiently seeks a feasible primitive combination using Bayesian optimization, by reducing the number of observations required.

3 Sequential Assembly with Unit Primitives

In this section, we provide the detailed configurations and assumptions, which are used to propose our combinatorial 3D shape generation method. Volumetric representation of our interest is the most straightforward expression to define interactions between primitives and assemble a shape with a set of geometric primitives. However, the choice for the types of unit geometric primitives is important, because it makes this sequential assembly either too simple or too challenging.

Assume that we have a target shape (i.e., Figure 2(a)) to assemble. If we use 1×1-sized voxel primitives, there exists only *one combination* composed of 24 primitives, as shown in Figure 2(b). However, if we use the primitives identical to 2×4 volumetric primitives, we can assemble *two combinations* (i.e., Figure 2(c)). These observations indicate that smaller and simpler primitives tend to create more fine-grained shapes but have less combinatorial sequences. On the contrary, larger and more complicated primitives tend to create coarser shapes but have more interesting combinatorial sequences. These facts also relates to the real-world problem when we realize shapes by assembling multiple parts. As a result, we restrict the primitive we employ as a single type, which can only be connected with another primitive in fixed configurations and does not allow overlap between primitives. In this work, we select a 2×4 LEGO brick as a unit primitive, since a 2×4-sized primitive with studs and cavities is one of the representative basic LEGO bricks.[2]

4 Occupiability Grid

To tackle an assembly problem, we start by defining a search region $\mathcal{S}\subseteq\mathbb{R}^3$, which is a space to construct a 3D object. We employ *occupiability grid*, which is the opposite concept of probabilistic volumetric models [28]. The occupiability grid is a grid of which the unit cell (i.e., a voxel) possesses the possibility of being occupied in the future.

Given the number of partitions for each of three axes m_1, m_2, and m_3, a voxel $\mathbf{v}_{ijk}$ can be represented as (i,j,k) where $i\in[m_1]$, $j\in[m_2]$, and $k\in[m_3]$, and a collection of entire voxels is $\{\mathbf{v}_{ijk}\}_{i\in[m_1],j\in[m_2],k\in[m_3]}$. For a generic voxel grid, the occupancy of a voxel is expressed as one of two alternatives:

$$q(\mathbf{v}_{ijk})=\begin{cases}1 & \text{if it is occupied,}\\ 0 & \text{otherwise.}\end{cases} \tag{1}$$

On the other hand, the occupiability of voxel $\mathbf{v}_{ijk}$ can be expressed as

$$o(\mathbf{v}_{ijk})=\begin{cases}1 & \text{if it is occupiable,}\\ 0 & \text{otherwise.}\end{cases} \tag{2}$$

[2]We also have a historical reason for this choice. As introduced in [15], a 2×4 LEGO brick is the first patent granted in 1947 of the LEGO company and it opens a progressive development of building a 3D shape.

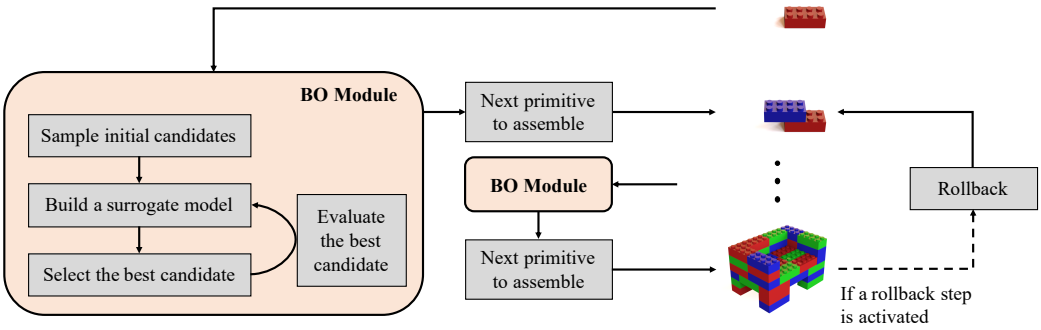


Figure 3: Overview of our combinatorial 3D shape generation pipeline.

In contrast with voxel definition, our concept of occupiability is to capture the likelihood of an empty voxel would be occupied in the near future. Thus, the voxel that is already occupied or prohibited due to specific constraints will be regarded as 0 occupiability.

Since a volumetric 2×4 brick as a unit primitive is placed in $\mathcal{S}$, it should transform into a single covariate to compare to other primitives. We thus denote the coordinate of each primitive as a 3D vector $\mathbf{x} = [x_1, x_2, x_3] \in \mathcal{S}$, where (x_1, x_2) is the center over the first two axes and x_3 is the bottom of the primitive, and the direction of each primitive as a scalar d. With this representation, suppose that every primitive is placed over the plane that $x_3 = 0$, and d of each primitive is placed either lengthwise (i.e., denoted as $d = 0$) or breadthwise (i.e., denoted as $d = 1$). It means every primitive can turn either 0, $\pi/2$, π, or $3\pi/2$ radians along the third axis of $\mathcal{S}$. To sum up, each primitive is defined as a tuple $(\mathbf{x}, d)$ (henceforth, denoted as $\mathbf{p} \in \mathcal{P}$ where $\mathcal{P} = \mathcal{S} \times \{0, 1\}$) where $x_3 \geq 0$ and $d \in \{0, 1\}$, and a n-primitive combination is expressed as a set $\{\mathbf{p}_i\}_{i=1}^n = \{(\mathbf{x}_i, d_i)\}_{i=1}^n$.

5 Combinatorial 3D Shape Generation

An assembling sequence is generated where each step in the series is suggested by one of the sequential model-based optimization methods, Bayesian optimization [6]. The Bayesian optimization strategy efficiently samples the position of the next primitive to assemble.

Evaluating Primitive Combinations To determine the position of the next primitive guided by Bayesian optimization, we need to define two evaluation functions over $(n+1)$-primitive combinations. One f_o is related to occupiability and the other f_s is related to stability:

$$y_\text{o} = f_\text{o}\left(\mathbf{p}_{n+1}; \{\mathbf{p}_l\}_{l=1}^n\right) + \epsilon_\text{o} \quad \text{and} \quad y_\text{s} = f_\text{s}\left(\mathbf{p}_{n+1}; \{\mathbf{p}_l\}_{l=1}^n\right) + \epsilon_\text{s}, \tag{3}$$

which are considered as true functions, where ϵ_o and ϵ_s are observation noises.

We design f_o to guide to follow a global shape context without providing exact probable positions of the primitive we would like to place. f_s is to measure time to be stable where gravity is existed and external forces at the beginning are applied. Suppose that $\mathbf{v}_{ijk}$ is a voxel of our interest and $\{\mathbf{p}_l\}_{l=1}^n$ is the current primitive combination. Given a desired shape $\{\mathbf{v}_l\}_{l=1}^N$, the occupiability is the possibility to be occupied:

$$o(\mathbf{v}_{ijk}; \{\mathbf{p}_l\}_{l=1}^n) = q\left(\mathbf{v}_{ijk}; \{\mathbf{v}_l\}_{l=1}^N\right) \wedge \left(1 - q\left(\mathbf{v}_{ijk}; \{\mathbf{p}_l\}_{l=1}^n\right)\right). \tag{4}$$

With Eq. (3) and Eq. (4), the occupiability score over $\mathbf{p}_{n+1}$ is

$$f_\text{o}\left(\mathbf{p}_{n+1}; \{\mathbf{p}_l\}_{l=1}^n\right) = \sum_{\mathbf{v}_{ijk} \in \mathbf{p}_{n+1}} o(\mathbf{v}_{ijk}; \{\mathbf{p}_l\}_{l=1}^n). \tag{5}$$

As the other evaluation function, we measure time to be stable where gravity is constantly applied to the combination and external forces are applied at the beginning of the simulation. Since measuring time to be stable in practice is a time-consuming and difficult task, we test an artificial experiment on

Algorithm 1 Find a Feasible Primitive Combination

Input: An initial primitive combination $\{\mathbf{p}_i\}_{i=1}^{m}$, the number of primitives to assemble T, the number of rollback steps τ, threshold for rolling back α.

Output: A primitive combination $\{\mathbf{p}_j\}_{j=1}^{m+T}$.

Initialize $t \leftarrow 0$ and compute $y_{\mathrm{o},0}$, $y_{\mathrm{s},0}$ with $\{\mathbf{p}_t\}_{i=1}^{m}$.
while $t < T$ **do**
  Select a next primitive to assemble $\mathbf{p}_{t+1}$, using Algorithm 3.
  Assemble $\mathbf{p}_{t+1}$ to the combination, $\{\mathbf{p}_t\}_{i=1}^{m+t}$.
  Update $t \leftarrow t + 1$.
  Compute $y_{\mathrm{o},t}$, $y_{\mathrm{s},t}$ with $\{\mathbf{p}_t\}_{i=1}^{m+t}$.
  if $t \geq \tau$ and $\sum_{k=0}^{\tau-1} \left(\max_{\mathbf{p}_{m+t-k}} y_{\mathrm{o},t-k}\right) - y_{\mathrm{o},t-k} < \alpha$ **then**
    Roll back $\{\mathbf{p}_t\}_{i=1}^{m+t}$ to $\{\mathbf{p}_t\}_{i=1}^{m+t'}$ where $t' = t - \tau$.
    Update $t \leftarrow t - \tau$.
  end if
end while
Return $\{\mathbf{p}_j\}_{j=1}^{m+T}$

Algorithm 2 Query a Candidate of Next Primitive

Input: A n-primitive combination $\{\mathbf{p}_i\}_{i=1}^{n}$, r possible primitives observed and their evaluation scores with the n-primitive combination $\{(\mathbf{p}_j, s_j)\}_{j=1}^{r}$, the number of samples for acquisition function optimization ζ.

Output: A candidate of next primitive $\mathbf{p}_{r+1}^{*}$.

Estimate a surrogate function:

$$\hat{f}\left(\mathbf{p}; \{\mathbf{p}_i\}_{i=1}^{n}, \{(\mathbf{p}_j, s_j)\}_{j=1}^{r}\right),$$

using the primitive combination and the historical observations.
Sample ζ primitives possible to assemble.
Find a maximizer $\mathbf{p}_{r+1}^{*}$, i.e., one of ζ primitives, of the acquisition function computed by $\hat{f}$.
Return $\mathbf{p}_{r+1}^{*}$

a physics engine simulator. To precisely measure the stability where instability is implanted, we apply the external forces to one of four directions alternately at the beginning, and then measure time steps until the position of combinations is not changed: $f_{\mathrm{s}}\left(\mathbf{p}_{n+1}; \{\mathbf{p}_l\}_{l=1}^{n}\right) = \tau$, if $\frac{1}{n+1}\sum_{l=1}^{n+1} \|\mathbf{p}_l^{(\tau)} - \mathbf{p}_l^{(\tau-1)}\|_2 < \varepsilon$, where $\mathbf{p}^{(\tau)}$ indicates the primitive position at a time step τ, and ε is a threshold for terminating the simulation. The details are described in Section 6 and the appendices.

Determining the Next Primitive to Assemble Using the aforementioned evaluation functions over primitive combinations, we can efficiently determine the next position to assemble. Bayesian optimization, which is a sample-efficient global optimization method for black-box functions, sequentially finds the next primitive candidate that maximizes an acquisition function. The benefit of utilizing Bayesian optimization is that we do not need to assume differentiability, continuity, or any other specific functional form of the original function [6, 32].

As shown in Eq. (3), the evaluation functions that define where we should assemble cannot be optimized using generic optimization strategies due to the unknown of functional forms. For this property, Bayesian optimization is utilized to decide where a primitive should be assembled without human intervention. Moreover, determining a primitive position to assemble is taken into account as a process to reveal where we assemble the next position among huge possible primitive positions, which is a sequential combinatorial procedure to assembling primitives with Bayesian optimization. In this intuition, Algorithm 1, Algorithm 2, and Algorithm 3 are introduced.

First of all, similar to common Bayesian optimization [19, 25], a surrogate function over primitives is built given r historical observations $\{(\mathbf{p}_i, y_{\mathrm{o},i}, y_{\mathrm{s},i})\}_{i=1}^{r}$. Commonly, Gaussian process regression [29] is used as a surrogate function in the Bayesian optimization community [6, 33], because it can express any function in the reproducing kernel Hilbert space. Note that each primitive $\mathbf{p}$ is regarded as a

Algorithm 3 Select the Next Primitive Position to Assemble

Input: Initial primitive combination $\{\mathbf{p}_i\}_{i=1}^m$, the number of initial primitives v, the number of primitive candidates $q > v$.
Output: The next primitive to assemble $\mathbf{p}_{m+1}$.
1: Sample v primitives to be able to assemble randomly, $\{\mathbf{p}_i\}_{i=1}^v$.
2: Evaluate the primitive combinations each of which is composed of initial combination $\{\mathbf{p}_i\}_{i=1}^m$ and one of v primitives, $\{(\mathbf{p}_i, y_{\mathrm{o},i}, y_{\mathrm{s},i})\}_{i=1}^v$.
3: **for** $j = v+1, \ldots, q$ **do**
4: Query the next primitive candidate to assemble $\mathbf{p}_j$.
5: Evaluate the primitive combination composed of $\{\mathbf{p}_i\}_{i=1}^m$ and $\mathbf{p}_j$.
6: Update the candidate set, $\{(\mathbf{p}_i, y_{\mathrm{o},i}, y_{\mathrm{s},i})\}_{i=1}^j$.
7: **end for**
8: Select the next primitive $\mathbf{p}_{m+1}$, which has achieved the best score from $\{(\mathbf{p}_i, y_{\mathrm{o},i}, y_{\mathrm{s},i})\}_{i=1}^q$.
9: **Return** $\mathbf{p}_{m+1}$

four-dimensional vector, composed of a 3D vector of primitive $\mathbf{x}$ and a direction of primitive d. By the Gaussian process regression, given r four-dimensional inputs $\mathbf{P} = [\mathbf{p}_1 \cdots \mathbf{p}_r]^\top \in \mathbb{R}^{r\times 4}$, their associated outputs for occupiability $\mathbf{y}_\mathrm{o} = [y_{\mathrm{o},1} \cdots y_{\mathrm{o},r}] \in \mathbb{R}^r$, and their associated outputs for stability $\mathbf{y}_\mathrm{s} = [y_{\mathrm{s},1} \cdots y_{\mathrm{s},r}] \in \mathbb{R}^r$, a function value and its uncertainty are represented by the posterior mean and variance functions for occupiability or stability: $\mu_j(\mathbf{p}) = \mathbf{k}(\mathbf{p}, \mathbf{P}) \left(\mathbf{K}(\mathbf{P}, \mathbf{P}) + \sigma_n^2 \boldsymbol{I}\right)^{-1} \mathbf{y}_j$ and $\sigma_j^2(\mathbf{p}) = k(\mathbf{p}, \mathbf{p}) - \mathbf{k}(\mathbf{p}, \mathbf{P}) \left(\mathbf{K}(\mathbf{P}, \mathbf{P}) + \sigma_n^2 \boldsymbol{I}\right)^{-1} \mathbf{k}(\mathbf{P}, \mathbf{p})$, where $j \in \{\mathrm{o}, \mathrm{s}\}$, and the covariance functions are defined as $k : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}$, $\mathbf{k} : \mathbb{R}^4 \times \mathbb{R}^{r\times 4} \to \mathbb{R}^r$, and $\mathbf{K} : \mathbb{R}^{r_1 \times 4} \times \mathbb{R}^{r_2 \times 4} \to \mathbb{R}^{r_1 \times r_2}$. In addition, σ_n^2 is a noise scale and $\boldsymbol{I}$ is an identity matrix.

We compute the acquisition function values for ζ primitives possible to assemble, and find a maximizer among the ζ primitives. To cope with two evaluation functions, we employ a multi-objective Bayesian optimization with random scalarizations [27]. A maximizer $\mathbf{p}^*$ of the acquisition function over $\mathbf{p}$ is found to observe the true functions:

$$\mathbf{p}^* = \underset{\mathbf{p}\in\mathcal{P}}{\arg\max}\, \lambda_\mathrm{o} a(\mathbf{p}; \mu_\mathrm{o}(\mathbf{p}), \sigma_\mathrm{o}^2(\mathbf{p})) + \lambda_\mathrm{s} a(\mathbf{p}; \mu_\mathrm{s}(\mathbf{p}), \sigma_\mathrm{s}^2(\mathbf{p})), \tag{6}$$

where $\mathcal{P}$ is a compact set, and λ_o, λ_s are random weights sampled from uniform distributions. As an acquisition function, in this paper, we use Gaussian process upper confidence bound (GP-UCB) [34], $a_{\mathrm{UCB}}(\mathbf{p}; \mathbf{P}, \mathbf{s}) = \mu(\mathbf{p}) + \gamma\sigma(\mathbf{p})$, where γ is a trade-off hyperparameter for exploitation and exploration. This procedure is presented in Algorithm 2.

Sampling ζ is the main difference between our method and common Bayesian optimization strategies. Well-known techniques for optimizing an acquisition function (e.g., DIRECT [18] and L-BFGS-B [23]) poorly work, because our search space contains the complicated constraints that are basically determined by occupiabilities. Besides, the combinatorial approach [2] is difficult to apply due to the curse of dimensionality, which is derived from the combinatorial explosion of inputs. We thus sample a feasible set from a primitive set, each element of which can assemble. This technique is used in the automated machine learning community [16].

As shown in Algorithm 3, after choosing v initial random primitives and evaluating those primitives with $\{\mathbf{p}_i\}_{i=1}^m$, where m is the cardinality of given primitive combination, a primitive candidate is queried, and new observation is accumulated, until q primitives are observed. Finally, the next primitive that has achieved the best score for occupiability is returned. Algorithm 1 describes how a feasible primitive combination is assembled with the evaluation functions that guide an assembling process. Consequently, we obtain a $(m+T)$-primitive combination.

Rolling Back the Primitives Previously Assembled Our method might be able to provide a sub-optimal sequence, because the Bayesian optimization module, which always guarantees to find a global solution [6, 34], accumulates local (but possibly global) optima in a row. Hence, our method includes a rollback step, as shown in Line 7 to Line 10 in Algorithm 1. Given the number of rollback steps τ and a threshold for rolling back α, if the condition written below is satisfied: $t \geq \tau$ and $\sum_{k=0}^{\tau-1} \left(\max_{\mathbf{p}_{m+t-k}} y_{\mathrm{o},t-k}\right) - y_{\mathrm{o},t-k} < \alpha$, t-primitive combination is rolled back to $(t-\tau)$-primitive combination. To avoid rolling back in the same combination infinitely, we prevent placing the same

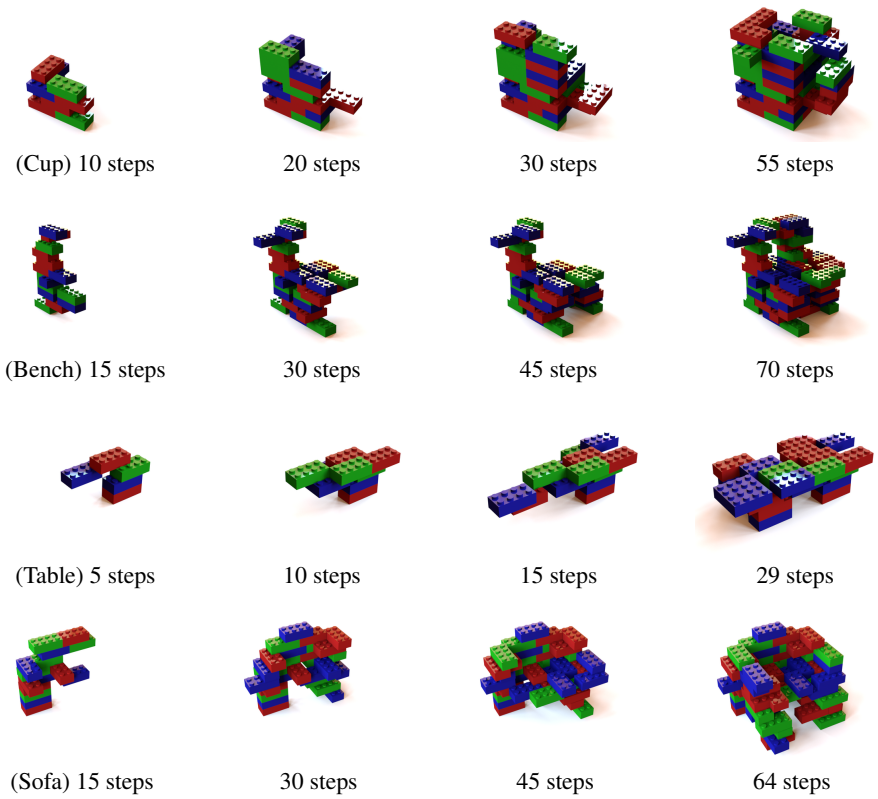
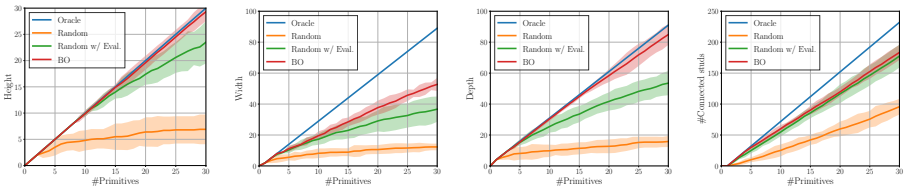


Figure 4: Results on sequential primitive assemblies.



(a) Height (b) Width (c) Depth (d) #Connected studs

Figure 5: Quantitative results on maximizing four explicit evaluation functions (i.e., height, width, depth, and the number of connected studs). BO indicates our sequential assembly method. All the experiments are repeated ten times, and their means and the 1.96 standard deviations are plotted.

position if the rollback is activated and we skip the rollback if the same assembling step is repeated for five times.

6 Experimental Results

Combinatorial 3D Shape Generation via Sequential Assembly We sequentially generate a combinatorial 3D shape by optimizing Eq. (3), given the occupiability with a desired shape, as described in the main method section. Our assembly results shown in Figure 1 and Figure 4 demonstrate that our method can effectively find a feasible combination, by considering our evaluation functions. To accommodate a page limit, we provide additional experimental results in the appendices.

Explicit Evaluation Functions for Bayesian Optimization Module To confirm the validity of our Bayesian optimization module where an evident evaluation function is given. For example, the

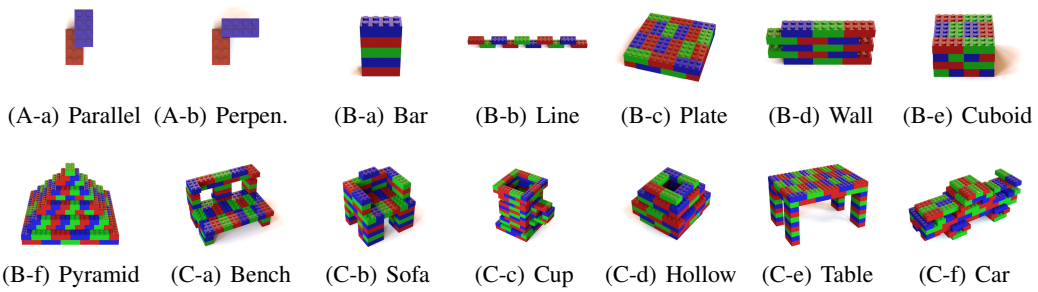


Figure 6: Selected examples from our combinatorial 3D shape dataset.

height of combination can express the current status of the given combination as an evaluation score:

$$f\left(\mathbf{p}_{n+1}; \{\mathbf{p}_i\}_{i=1}^{n}\right) = \max_{(x_1,x_2,x_3,d)\in\{\mathbf{p}_i\}_{i=1}^{n+1}} x_3 + 1. \tag{7}$$

Similar to Eq. (7), we define three more specific functions: width, depth, and the number of connected studs. Likewise, these BO modules attempt to maximize the evaluation functions we define. For these experiments, we use three baselines, which are described in the appendices. As shown in Figure 5, our method outperforms other methods and achieves the results closest to the oracles of four experiments.

7 Combinatorial 3D Shape Dataset

We construct a combinatorial 3D shape dataset, composed of 406 instances of 14 classes. Specifically, each object in our dataset is considered equivalent to a sequence of primitive placement. For this reason, compared to other 3D object datasets [7, 40], our proposed dataset contains an assembling sequence of unit primitives. It implies that we can quickly obtain a sequential generation process that is a human assembling mechanism. Furthermore, we can sample valid random sequences from a given combinatorial shape after validating the sampled sequences. To sum up, the characteristics of our combinatorial 3D shape dataset are

(i) combinatorial: Duplicates of unit primitive is repeatedly connected;

(ii) sequential: Allowable connections between primitives are sequentially added;

(iii) decomposable: By the combinatorial property, parts of combination can be sampled if they are valid in terms of the contact and overlap conditions;

(iv) manipulable: New primitive is addable or the existing primitives are removable.

Our 3D shape dataset is implemented to satisfy the above properties, supporting sequential assembly, combination validation, possible position listing, and part sampling.

As shown in Figure 6, we select 14 classes: parallel, perpendicular, bar, line, plate, wall, cuboid, square pyramid, bench, sofa, cup, hollow, table, and car. Parallel that implies the directions of two primitives are same, and perpendicular that implies the directions of two primitives are different classes are own connection types of 2×4-sized primitives with studs and cavities (denoted as group A). Bar, plate, cuboid, wall, and square pyramid classes are taken into account as default components to assemble sophisticated shapes (denoted as group B). The other classes are abstractly thought of as the combination of those default components (denoted as group C). More diverse examples and the statistics of our dataset can be found in the appendices.

8 Conclusion

We propose a sequential assembly method for a combinatorial 3D generation problem. It can generate a combinatorial shape, considering sample efficiency that is guided by Bayesian optimization. The evaluation function based on global shape guidance and stability demonstrates that our method generates 3D shapes composed of unit primitives. Besides, we create a new dataset for combinatorial 3D models. This dataset allows us to generate 3D shapes sequentially.

Acknowledgments and Disclosure of Funding

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.2019-0-01906, Artificial Intelligence Graduate School Program (POSTECH)).

References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3D point clouds. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[2] R. Baptista and M. Poloczek. Bayesian optimization of combinatorial structures. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[3] M. P. Bendsøe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71(2): 197–224, 1988.

[4] T. Borrvall and J. Petersson. Topology optimization of fluids in Stokes flow. *International Journal for Numerical Methods in Fluids*, 41(1):77–107, 2003.

[5] D. Brackett, I. Ashcroft, and R. Hague. Topology optimization for additive manufacturing. In *Proceedings of the Solid Freeform Fabrication Symposium*, 2011.

[6] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[7] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[8] E. Coumans and Y. Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[9] S. Eilers. The LEGO counting problem. *The American Mathematical Monthly*, 123(5):415–426, 2016.

[10] H. A. Eschenauer, V. V. Kobelev, and A. Schumacher. Bubble method for topology and shape optimization of structures. *Structural Optimization*, 8(1):42–51, 1994.

[11] S. M. Felton, M. T. Tolley, C. D. Onal, D. Rus, and R. J. Wood. Robot self-assembly by folding: A printed inchworm robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[12] L. Gao, J. Yang, T. Wu, Y. Yuan, H. Fu, Y. Lai, and H. Zhang. SDM-NET: Deep generative network for structured deformable mesh. *arXiv preprint arXiv:1908.04520*, 2019.

[13] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. AtlasNet: A Papier-Mâché approach to learning 3D surface generation. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[14] J. D. Hartgerink, E. Beniash, and S. I. Stupp. Self-assembly and mineralization of peptide-amphiphile nanofibers. *Science*, 294(5547):1684–1688, 2001.

[15] S. Herman. *Building a History: The LEGO Group*. Grub Street Publishers, 2012.

[16] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*, 2011.

[17] W. Jakob. Mitsuba renderer. `http://www.mitsuba-renderer.org`, 2010.

[18] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[19] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[20] G. Kharmanda, N. Olhoff, A. Mohamed, and M. Lemaire. Reliability-based topology optimization. *Structural and Multidisciplinary Optimization*, 26(5):295–307, 2004.

[21] J. Kim and S. Choi. BayesO: A Bayesian optimization framework in Python. `http://bayeso.org`, 2017.

[22] S. Lee, J. Kim, J. W. Kim, and B. Moon. Finding an optimal LEGO® brick layout of voxelized 3D object using a genetic algorithm. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO)*, 2015.

[23] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.

[24] S. Luo, Y. Yue, C. Huang, Y. Chung, S. Imai, T. Nishita, and B. Chen. Legolization: Optimizing LEGO designs. *ACM Transactions on Graphics*, 34(6):222:1–222:12, 2015.

[25] J. Moćkus, V. Tiesis, and A. Źilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129, 1978.

[26] C. Nash, Y. Ganin, S. M. A. Eslami, and P. W. Battaglia. PolyGen: An autoregressive generative model of 3D meshes. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[27] B. Paria, K. Kandasamy, and B. Póczos. A flexible framework for multi-objective Bayesian optimization using random scalarizations. In *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2019.

[28] T. Pollard and J. L. Mundy. Change detection in a 3-d world. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[29] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[30] J. W. Romanishin, K. Gilpin, and D. Rus. M-blocks: Momentum-driven, magnetic modular robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[31] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[32] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[33] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[34] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2010.

[35] R. Testuz, Y. Schwartzburg, and M. Pauly. Automatic generation of constructable brick sculptures. In *Proceedings of the Annual Conference of the European Association for Computer Graphics (Eurographics)*, 2013. Short Paper.

[36] H. Wei, Y. Cai, H. Li, D. Li, and T. Wang. Sambot: A self-assembly modular robot for swarm robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[37] G. M. Whitesides and B. Grzybowski. Self-assembly at all scales. *Science*, 295(5564):2418–2421, 2002.

[38] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[39] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[40] Y. Xiang, W. Kim, W. Chen, J. Ji, C. Choy, H. Su, R. Mottaghi, L. Guibas, and S. Savarese. ObjectNet3D: A large scale database for 3D object recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[41] Q. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv preprint arXiv:1801.09847*, 2018.

Appendices

We describe the details which are omitted from the main article. First, we show the connection types between two 2×4-sized primitives. Then, we describe the experimental setups and visualize some examples of our combinatorial 3D shape dataset.

Appendix A Connection Types between Two Two-by-Four Primitives

There exist 46 connection types between two 2×4 LEGO brick-shaped primitives where upper and lower primitives are fixed, as shown in Figure 7. They comprise group A of our combinatorial dataset.

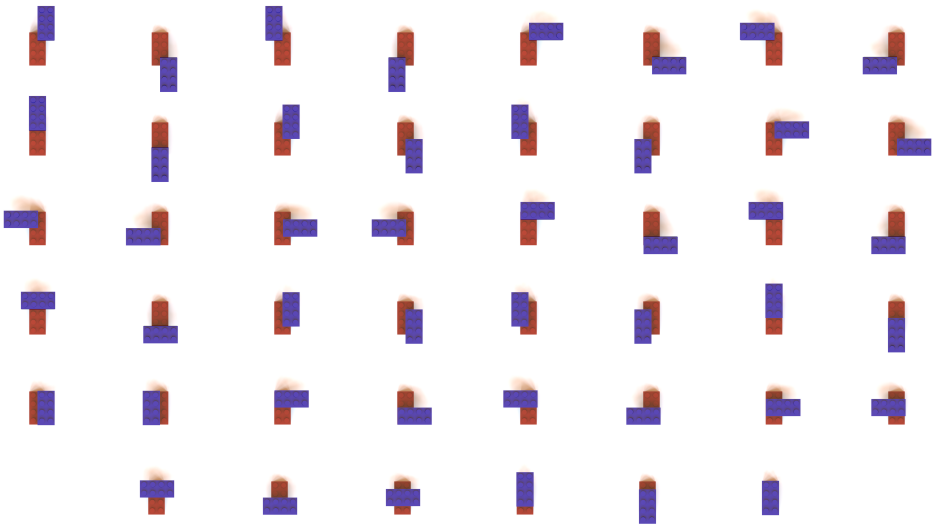
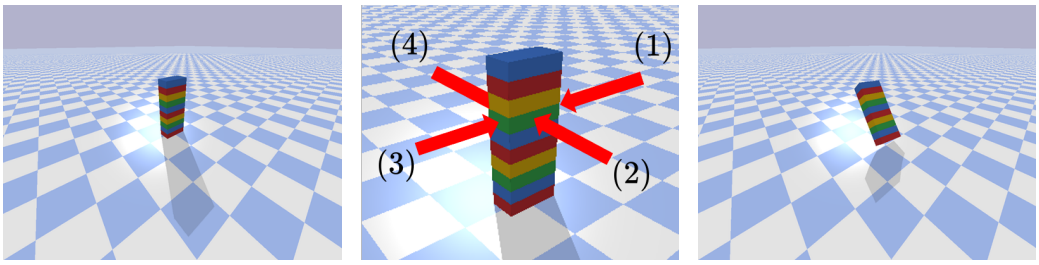


Figure 7: 46 connection types between two 2×4-sized primitives.

Appendix B Experimental Setups

Gaussian process regression with Matérn 5/2 kernels [29] and Gaussian process upper confidence bound (GP-UCB) [34] are used as a surrogate model and an acquisition function, respectively. The hyperparameters of kernels for the regression models are optimized by marginal likelihood maximization with BFGS algorithm. Similar to the setting in [34], the trade-off hyperparameter of GP-UCB monotonically increases over iterations. Rather than setting a specific number of samples ζ for



(a) Placing (b) Applying forces (c) Measuring stability

Figure 8: Stability simulation results with `PyBullet`.

Line 4 of Algorithm 3, we sample as many candidates of maximizer as possible within the given time budget. We set the time budget as 1 second in most of the cases. Unless otherwise specified, v and q in Algorithm 3 are set to 10 and 20, respectively. In all the experiments, an initial primitive combination in Algorithm 1 is given as $\{([0, 0, 0], 0)\}$, to assemble from scratch. Bayesian optimization modules are implemented with [21]. For multi-objective Bayesian optimization, λ_{o} is sampled from uniform distribution $\mathcal{U}(0.8, 0.9)$ and λ_{s} is sampled from uniform distribution $\mathcal{U}(0.0, 0.1)$.

To measure the stability, we use the physics engine simulator [8] as shown in Figure 8. We first place a combination we would like to measure the stability. And then, we alternately apply external forces to one of four directions (see Figure 8(b)) at the beginning of the simulation (i.e., 200 time steps) where gravity is existed. After applying the forces, we measure time steps to be stable in terms of the position of given combination.

Open3D framework [41] and Mitsuba renderer [17] are employed to deal with 3D objects and visualize the primitive assembly results. For attractive visualization, we randomly pick the color of primitives from red, blue, and green.

For the experiments for explicit evaluation functions, we compare our method to three baselines:

(i) Oracle: It is the best achievable result;

(ii) Random: This baseline is a fully-randomized result. One of the primitives possible to assemble is uniformly selected at every assembling step;

(iii) Random with evaluations: It is a result by a greedy method. The best primitive is chosen at every step after evaluating the primitives uniformly sampled. The number of the sampled primitives is set to q to compare with our method fairly.

Appendix C Combinatorial 3D Shape Dataset

We demonstrate a part of our combinatorial 3D shape dataset, as shown in Figure 7, and Figure 9 to Figure 20. In addition, the statistics on our dataset is specified in Table 1.

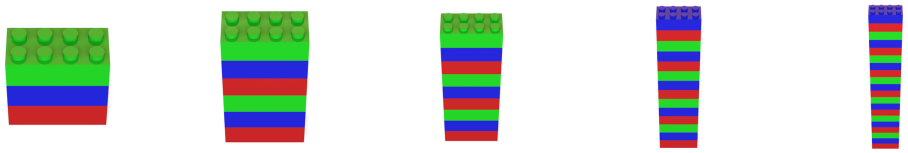


Figure 9: Examples of *Bar* class.

Figure 10: Examples of *Line* class.

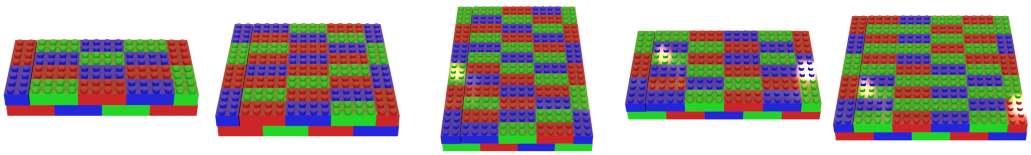


Figure 11: Examples of *Plate* class.

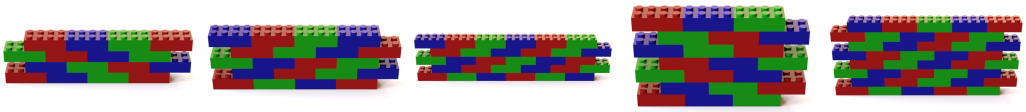


Figure 12: Examples of *Wall* class.

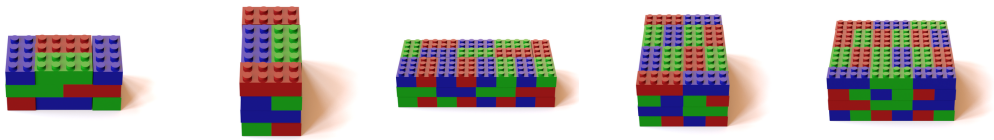


Figure 13: Examples of *Cuboid* class.

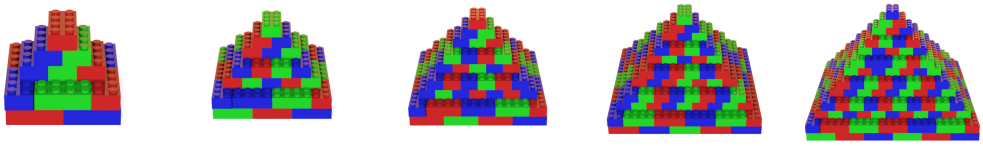


Figure 14: Examples of *Square Pyramid* class.

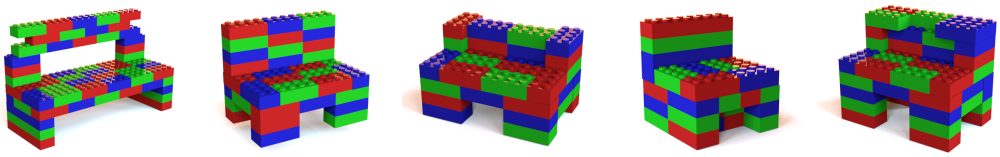


Figure 15: Examples of *Bench* class.

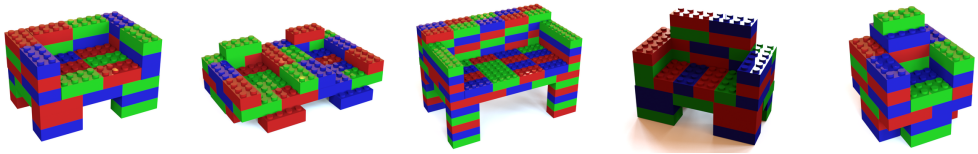


Figure 16: Examples of *Sofa* class.

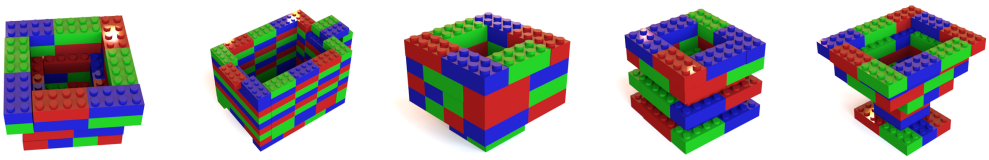


Figure 17: Examples of *Cup* class.

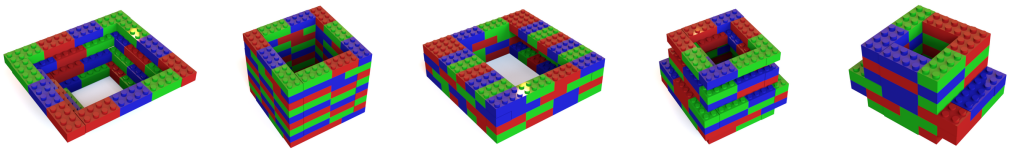


Figure 18: Examples of *Hollow* class.

Figure 19: Examples of *Table* class.

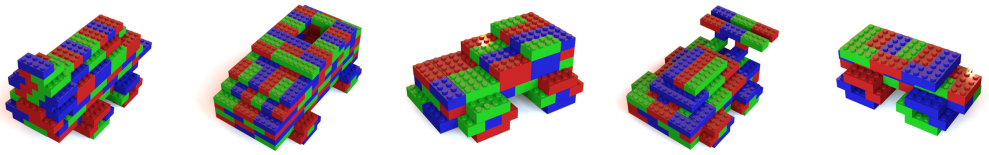


Figure 20: Examples of *Car* class.

Table 1: Statistics on combinatorial 3D shape dataset. Std stands for standard deviation.

Group A	Parallel	Perpen.				
#instances	21	25				
Mean of #primitives per instance	2.0	2.0				
Std of #primitives per instance	0.0	0.0				
Group B	Bar	Line	Plate	Wall	Cuboid	Sq. Pyramid
#instances	30	30	30	30	30	30
Mean of #primitives per instance	11.9	32.5	56.0	27.9	26.4	164.0
Std of #primitives per instance	6.6	42.0	35.1	14.6	17.6	129.2
Group C	Bench	Sofa	Cup	Hollow	Table	Car
#instances	30	30	30	30	30	30
Mean of #primitives per instance	55.4	59.6	49.7	46.3	36.9	83.6
Std of #primitives per instance	28.0	30.5	31.2	31.8	19.2	41.0