ShapeCrafter: A Recursive Text-Conditioned 3D Shape Generation Model

Rao Fu **Xiao Zhan** **Yiwen Chen** **Daniel Ritchie** **Srinath Sridhar**

Brown University
rao_fu@brown.edu
ivl.cs.brown.edu//projects/shapecrafter

Abstract

We present **ShapeCrafter**, a neural network for recursive text-conditioned 3D shape generation. Existing methods that generate text-conditioned 3D shapes consume an entire text prompt to generate a 3D shape in a single step. However, humans tend to describe shapes recursively—we may start with an initial description and progressively add details based on intermediate results. To capture this recursive process, we introduce a method to generate a 3D shape distribution, conditioned on an initial phrase, that gradually evolves as more phrases are added. Since existing datasets are insufficient for training under this approach, we present **Text2Shape++**, a large dataset of 369K shape–text pairs that supports recursive shape generation. To capture local details that are often used to refine shape descriptions, we build upon vector-quantized deep implicit functions that generate a distribution of high-quality shapes. Results show that our method can generate shapes consistent with text descriptions, and shapes evolve gradually as more phrases are added. Our method supports shape editing, extrapolation, and can enable new applications in human–machine collaboration for creative design.

1 Introduction

Humans are unique in the animal kingdom in the use of language to describe objects, scenes and events [11]. We use language not only to describe an object's physical and functional attributes, but also to qualify or modify those attribute descriptions. More specifically, we use (linguistic) *recursion* [9]—we may start with an initial description of an object, and progressively add phrases to better describe it or modify it. We use this ability extensively, for instance when describing an object over the phone. The capability to understand recursive natural language descriptions of objects is crucial for the wide accessibility of intelligent robots, computational creativity tools, and online shopping systems.

Driven by recent advances in language modeling [14, 38, 5] and generative image modeling [24], there has been rapid progress in the problem of text-conditioned image generation [57, 46, 59, 60, 61]. For instance, methods such as DALL-E [41, 40] can generate photorealistic images from only text descriptions. This success in the 2D domain together with progress in 3D shape representations [34, 31, 33] has led to interest in text-conditioned *3D shape generation* [52, 42, 22]. However, existing methods have several shortcomings. First, these methods are limited to one-step shape generation and lack the ability to recursively modify or improve generated shapes [15]. Second, unlike in 2D images, there are no readily available large datasets of 3D shape–text pairs. Existing datasets are limited in size due to challenges in manually annotating 3D data and do not support recursive generation [7, 3]. Finally, current methods cannot capture local details that are commonly used in language descriptions.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

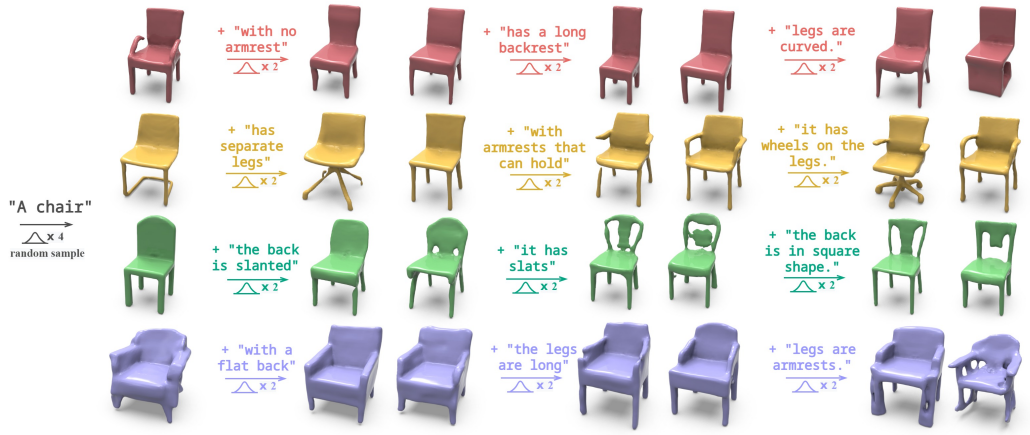


Figure 1: **ShapeCrafter** is a method for recursive text-conditioned 3D shape generation. Given an initial phrase (`A chair`), it generates a shape distribution (4 samples shown in the leftmost column). As more phrases are added, the initial shape is refined (2 samples shown). We can handle long phrase sequences while continuously evolving an initial shape (shape editing for a fixed random seed). Our method also shows extrapolation capabilities (`legs are armrests`, bottom right).

In this paper, we address the above limitations and present a method for **recursive text-conditioned generation of 3D shapes**. Our method, **ShapeCrafter**, consists of a transformer-based [51] neural network architecture that takes an initial text phrase (via BERT [14]) and generates a distribution of latent features over a 3D grid that can be sampled to obtain shapes consistent with the input. Our goal is to capture recursive phrase sequences common in language descriptions (see Figure 1)—as more informative words/phrases are added, our method produces more relevant shape distributions. To achieve this, we design a recursive network architecture that takes the latest phrase input in a sequence and incorporates latent grid features from the previous step to generate refined shapes. The shape distributions produced by our method evolve gradually as more phrases are added, thus enabling new applications in human–machine collaboration for creative design.

To address the limited size of existing datasets, we introduce **Text2Shape++**, a dataset that contains **369K** *many-to-many* mappings of varied-length phrase sequences that map to the same shape. This dataset is derived from Text2Shape [7] that contains 75K one-to-one text-shape pairs without recursive phrase sequences. We introduce a method to transform (see Section 4.1) the Text2Shape dataset into varied-length phrases resulting in a significantly larger dataset. Finally, to learn high-quality shapes, we adopt vector-quantized deep implicit functions (P-VQ-VAE) [58, 33] to better represent local details and generate shape distributions rather than single shapes.

Experiments and results produced by ShapeCrafter demonstrate that we can generate and evolve high-quality 3D shape distributions that are consistent with text inputs. As additional phrases are added to the sequence, the shapes gradually and continuously evolve while retaining shape details generated in the previous steps thus enabling **shape editing** functionality (see Figure 1). Our model can handle long phrase sequences that other methods cannot. Surprisingly, our model demonstrates the capability to extrapolate to novel inputs (*e.g.,* chair legs that look like armrests, see Figure 1 bottom right) unseen during training. Quantitative results show that our shape reconstruction quality is comparable to the state of the art, while our recursive shape generation and editing capabilities are state of the art. To sum up, our contributions are:

- **ShapeCrafter**, a neural network architecture that enables recursive text-conditioned generation of 3D shapes that continuously evolve as phrases are added.
- **Text2Shape++**, a new large dataset of **369K shape–text pairs** that can be used for recursive shape generation tasks.
- A model that captures local shape details corresponding to text descriptions and enables shape editing and extrapolation to novel text descriptions.

2 Related Work

In this brief review, we limit our discussion to work on 3D shape representations and 3D shape generation and manipulation. Text-conditioned 2D image generation [26, 41, 40] is outside of the scope of this review, please see [16, 19] for details.

3D Shape Representation and Generation: Existing work in 3D shape generation can be categorized based on their underlying representation of choice. Major choices include meshes [18, 56, 53, 17, 12], voxels [10, 28, 23, 7], point clouds [2, 27, 36, 54, 37, 43], neural fields/scene representation networks [32, 44], or deep implicit fields [34, 31, 8]. Meshes and point clouds are useful for simple shapes but struggle with complex details and topology resulting in lower quality results. Voxel grids take up a large amount of memory to preserve details. Implicit fields including SDFs are more popular because they can capture fine-grained details, handle arbitrary topologies, and produce watertight meshes. Since many neural implicits encode the whole shape into a single latent code, they often suffer from "posterior collapse", where the latent space is not fully used, and latents get ignored by the decoder [41, 50]. To combat this, the feature volume can be divided into smaller cells [45, 35]. More recently, vector quantized deep implicit functions (VQ-DIF) [58, 33] have gained popularity. VQ-DIFs encode shapes as series of discrete 2-tuples representing shape features and their position. This allows separation of specific features and effective re-combination in a transformer-based architecture. We adopt this approach in our work due to its high generation quality.

Text-Conditioned 3D Shape Generation: Text-conditioned 3D shape synthesis is significantly more challenging than text-conditioned image synthesis due to a lack of large text-shape datasets. Therefore, current methods use large pre-trained image–language embeddings like CLIP [38]. CLIP-conditional shape generation methods [52, 42, 22] associate encoded CLIP text features and CLIP image features obtained by rendering 3D shapes. Alternatively, sequence-to-sequence models [7, 3, 33, 30] are used to match texts and shapes. Unlike our method, none of the above methods can support long phrase sequences or recursive shape generation that controls local shape details.

Datasets: Training text-conditioned 3D shape generation models requires supervision in the form of text-shape pairs. Unlike in 2D, obtaining this labeled data is harder in 3D due to the complexity of creating text descriptions for large datasets like ShapeNet [6]. Nonetheless, Text2Shape [7] contains such data for two object classes (chairs, tables) in ShapeNet. In total, they provide 75K pairs (30K for chairs, 40K for tables) which is still insufficient for training models that capture local detail. ShapeGlot [3] provides a (2D) shape discrimination dataset with 94K rendered triplets for shape discrimination. None of these datasets support recursive shape learning. In our work, we provide a method to transform the Text2Shape dataset into a larger dataset of phrase sequence–shape pairs that supports recursive shape generation.

3 Background

We first introduce fundamental technical details and justification for our choice of shape representation, text feature extraction, and auto-regressive shape generation.

Shape Representation: Neural implicit representations [34] have been shown to capture shapes of arbitrary topology using global latent codes. However, single global latent codes can result in poor local details, so we use grid-based representations [35] which offer better local shape details. In this paper, we use a vector-quantized feature embedding space which represents each shape as a probability distribution of possible tokens in a fixed-size code book.

For each shape X, a shape encoder $E_\phi(\cdot)$ encodes its T-SDF (Truncated-Signed Distance Field) into a low dimensional 3D grid feature $E \in g^3 \times D$, where g is the resolution of the 3D grid, and D is the dimension of the 3D grid features. We define a discrete latent space which has K embedding vectors, where each of the vector has dimension D. For shape feature e_i at grid position i, a vector quantization operation $VQ(\cdot)$ is used to look up its the nearest neighbor e'_i from the K embedding vectors in the discrete latent space. The feature put into the shape decoder $D_\phi(\cdot)$ is the 3D discrete latent feature grid $E' \in g^3 \times D$, with discrete features e'_i at grid locations i. Therefore, we have:

$$E = E_\phi(X), E' = VQ(E), X' = D_\phi(E'), \tag{1}$$

where X' is the reconstructed shape, and ϕ is the set of learnable parameters of the decoder. As a result, a shape X can be represented as a 3D latent feature index grid $Q \in g^3$, where the number q_i at

each grid location corresponds to a discrete shape feature e'_i. To learn this, we use P-VQ-VAE [33], which extracts discrete latent features from volumetric T-SDFs. In our method, we empirically set $K = 512$, $D = 256$, and the 3D shape grid resolution is $g = 8$.

Text Feature Extraction: To extract semantic meaning from text inputs, we use BERT [14]. BERT is a bidirectional transformer pretrained on a large web-scale dataset that can be fine-tuned for different tasks. For each input text I, we use the first token in last layer of the $\mathbf{BERT_{BASE}}$ model as the text feature embedding $B \in \mathbb{R}^{768}$. In order to have all cells in the 3D feature grid to gather some semantic information from text input, we use a multilayer perceptron network $\Phi(\cdot)$ to project the text embedding feature such that it has the same resolution as the 3D feature grid. We have:

$$C = \Phi(B), \Phi : \mathbb{R}^{786} \rightarrow \mathbb{R}^{g^3}, \tag{2}$$

where C denotes the projected text features.

Autoregressive Generation: Both the 3D latent feature index grid Q extracted using P-VQ-VAE and the projected text feature C extracted by BERT are in the shape of a 3D grid. Features within the grid are correlated with each other. Our goal is to build a generation model that learns the joint-distribution of the grid while making the code in each grid dependent on previous phrase inputs.

To achieve this, we use a conditional autoregressive model for shape generation. Given grid-wise conditional feature $Z \in g^3 \times H$, where H is the dimension of the conditional feature (we set $H = 512$), the joint distribution of the 3D latent feature index grid Q and the conditional feature Z is formulated as $p(Q|Z) = \prod_{i=1}^{g3} p(q_i|q_{<i}, Z)$. We use the simplified assumption [33] that this joint distribution is a product of the shape prior, coupled with independent conditional terms Z:

$$p(q_i|q_{<i}, Z) = \prod_{i=1}^{g3} p_\theta(q_i|q_{<i}) \cdot p_\psi(q_i|Z), \tag{3}$$

where θ and ψ are trainable parameters.

4 Recursive Text-conditioned 3D Shape Generation

We now describe the details of how we achieve recursive text-conditioned shape generation. We first introduce our Text2Shape++ dataset in Section 4.1 since the dataset design influences our technical approach. Next, we introduce the "shape set" feature representation we use for recursive generation in Section 4.2. We then introduce the inference procedure of our method in Section 4.3. Finally, we introduce our training strategy that enables recursive generation in Section 4.4.

4.1 Text2Shape++: Dataset to Support Recursive Shape Generation

Our goal is to generate 3D shapes recursively from phrase input enabling the gradual evolution of generated 3D shapes. Unfortunately, no existing dataset can support this problem since text–shape pairs are limited to single phrases. Therefore, we construct a new dataset, Text2Shape++, building upon the Text2Shape [7] dataset, which has text–shape pairs of chair and table categories from ShapeNet [6]. Text2Shape++ contains **369K text–shape pairs**, which to our knowledge is the largest dataset of its kind. Each text prompt in the dataset is represented as a phrase sequence, and each phrase sequence corresponds to one or more shapes. On average, each phrase sequence corresponds to 16 models for the chair category, and 25 models for the table category.

Constructing Text2Shape++: Given each text prompt in the Text2Shape [7] dataset, we first split the text into sentences using a sentence splitter [21]. For each sentence, we use a constituency parser [25] to parse the sentence in to a constituency tree. There could be many phrase types in a constituency tree—we only consider 18 of them as valid phrase types that contain descriptions such as verbs, nouns, and adjectives. Meanwhile, we also mark the phrases that only contain stop-words[39] (*e.g.,* it, there, ...) as invalid. During the parsing procedure, if any child of the current node is not a valid phrase, we stop parsing. Thus, a sentence is parsed into a constituency tree, where all the leaf nodes are valid phrases that contain useful information. For a parsed tree Y, we use depth-first search to get all leaf nodes in the tree $I = [i_1, i_2, ..., i_T]$, where T denotes the number of leaves. We use I_t to denote the concatenation of the 1^{st} to the t^{th} phrases, and we refer to I_t as a phrase sequence.

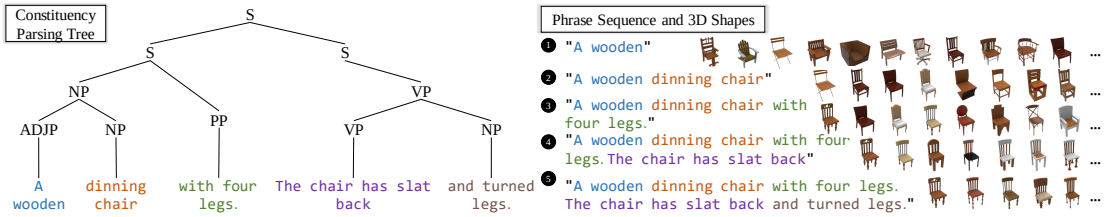


Figure 2: An example from Text2Shape++. Constituency parser [25] annotates a sentence with syntactic structure by decomposing it into phrases. Text2Shape++ contains phrase sequences, and each phrase sequence corresponds to one or more shapes.

To get shapes corresponding to phrase sequences, we calculate the similarity score of all the phrase sequences in our dataset with RoBERTa [29]. We consider two phrase sequences as *similar* if their similarity score is higher than a certain threshold (we empirically chose 0.94). For each phrase sequence, its corresponding shapes include the original Text2Shape dataset pair as well as shapes paired with its *similar* phrase sequences. Therefore, Text2Shape++ is a **many-to-many** text–shape correspondence dataset.

4.2 Shape Set Feature Representation.

Since any phrase sequences in Text2Shape++ can have many shapes (*i.e.,*, *shape set*) as their ground truth, we represent the shape set as a probability distribution of the discrete latent space rather a set of deterministic discrete latent feature codes. Recall that we use P-VQ-VAE [33] (see Section 3) to represent a 3D shape X as a 3D discrete latent feature index grid $Q \in g^3$. The discrete latent feature space has K latent features. Therefore, for a shape set that corresponds to a phrase sequence, it can be represented as a probability distribution of latent feature codes $Z \in g^3 \times K$, as illustrated in Fig. 3.

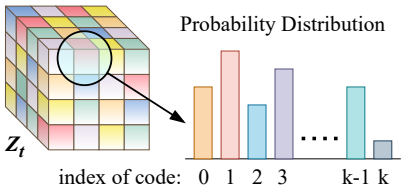


Figure 3: ShapeCrafter learns the probability distribution of latent features for each cell in Z.

Formally, each phrase sequence is provided with a shape set $\{X^j\}_{j=1}^M$ and their corresponding similarity scores $\{w^j\}_{j=1}^M$, where M denotes the number of shapes in the shape set. Given shape X^j represented as a 3D discrete latent feature index grid Q^j, its probability distribution of latent feature codes Z is deterministic, which can be represented as $Z_{ik} = \mathbb{I}\{k = Q_i\}$, where i denotes grid at position i, and k denotes the k^{th} dimension. Given a shape set $\{X^j\}_{j=1}^M$, where each shape X^j is represented by a 3D discrete latent feature index grid Q^j, we represent the shape set as a probability distribution by simply weighting the probability of the all the shapes with their similarity score:

$$Z^{set} = \frac{\sum_{j=1}^{M} w^j * Z^j}{\sum_{j=1}^{M} w^j}, \tag{4}$$

where Z^{set} is the shape representation of the shape set.

4.3 Recursive Shape Generation

We now describe the recursive inference procedure of ShapeCrafter which is also illustrated in Figure 4. ShapeCrafter uses the probability distribution of latent feature codes Z (Section 4.2) as its shape/shape set representation. To achieve text-conditioning, the phrase input is used to change the probability distribution of the latent feature code Z. As more phrases are added, the probabilities become narrower resulting in more deterministic shapes.

At time step t, ShapeCrafter takes two inputs: text input i_t from the current phrase, and the probability distribution of the latent feature code from the previous time step Z_{t-1}. At the first time step, we set Z_0 to be $\frac{1}{K}$, where $K = 512$ for each cell. The $\mathbf{BERT}_{BASE}$ model (Section 3) extracts text features B_t and the multi-layer perceptron network $\Phi(\cdot)$ projects the feature to C, which has the same

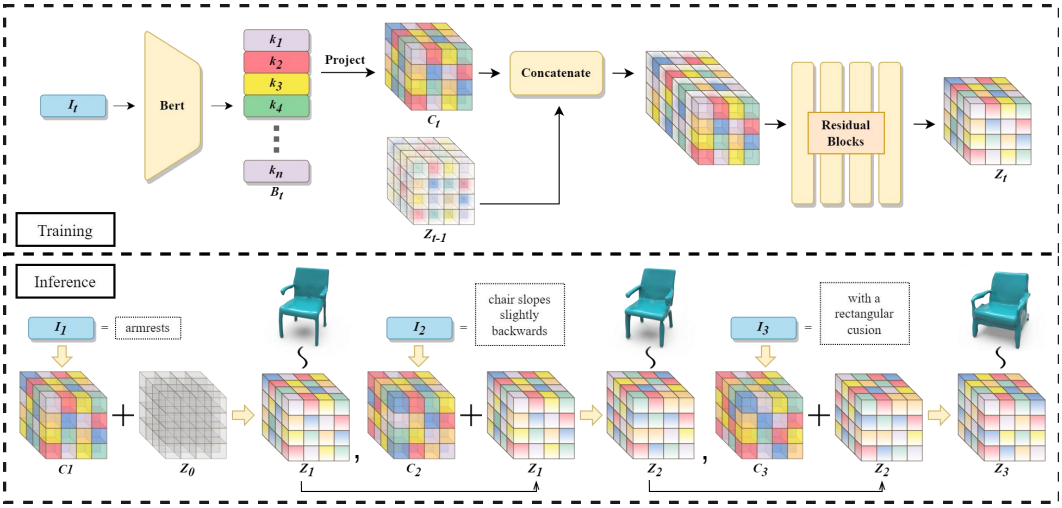


Figure 4: (Top) We take input text phrases and extract semantic features using BERT. These features are projected to a feature grid C_t which is concatenated with the latent feature code distribution Z_{t-1} from the previous time step. Residual blocks $\Psi(\cdot)$ output the feature grid distribution Z_t for the current time step. (2) During step t of inf1.2erence, we combine C_t and Z_{t-1} to obtain Z_t which is sampled to produce 3D shapes.

resolution as the 3D grid features. The text feature C is then concatenated with shape set feature Z_{t-1} along the channel dimension. Several residual blocks $\Psi(\cdot)$ are used to infer the probability distribution of the latent feature code at time t, which could be formulated as:

$$Z_t = \Psi([Z_{t-1}, C]), \tag{5}$$

where $[\cdot]$ stands for concatenation. Please see the supplementary document for more details.

The inferred probability distribution Z_t includes information from both text and shape at the previous step. However, the features at each grid are only weakly correlated. Therefore, Z_t is fed to the autoregressive generation model (Section 3) which generates the 3D discrete latent feature index grid Q. The VAE decoder D_ϕ introduced in Section 3 then decodes the index grid Q to a shape represented by a T-SDF.

At each recursive generation step, we want the generation model to prioritize distinct shape features from the current phrase at t. Therefore, we sort the sequence of inputs at the autoregressive step by the difference of the probability distribution at two time steps. We then use a random transformer [47] to generate the index grid sequentially, which could be formulated as:

$$p_t(q_{b_i}|q_{b_{<i}}, Z) = \prod_{i=1}^{N} p_{t\theta}(q_{b_i}|q_{b_{<i}}) \cdot p_{t\Psi}(q_{b_i}|Z), \{b_i\}_{i=1}^{N} = argsort(\|Z_t - Z_{t-1}\|_2^2), \tag{6}$$

where $N = g^3$ is the number of grids, and $argsort(\cdot)$ stands for descending sort.

4.4 Training

Five components are involved in training ShapeCrafter: (1) P-VQ-VAE model for shape representation; (2) Auto-regressive model for shape generation; (3) Fine-tuning $\mathbf{BERT}_{BASE}$ model for text feature extraction; (4) Text feature projection model $\Phi(\cdot)$; and (5) Residuals blocks $\Psi(\cdot)$ to extract the final distribution. We first follow the training strategy of AutoSDF [33] to train the P-VQ-VAE model and the autoregressive model, which are trained with both the feature of single shape and the feature of shape set. Then, we jointly train the $\mathbf{BERT}_{BASE}$ model, the projection model , and the residual blocks. Text2Shape++ provides us the dataset to train these three components since it provides phrase sequences $I = [i_1, i_2, ..., i_T]$ and the corresponding shape sets $\{X_t^j\}_{j=1}^M$. At time step t, we feed in phrase i_t and the shape set probability distribution feature Z_{t-1} corresponding to I_{t-1}. Note that Z_{t-1} is taken from Text2Shape++. We receive as output the predicted shape set probability

distribution feature $\hat{Z}_t$. Our loss function consists of reconstruction loss, vector quantization objective, and commitment loss as proposed by van den Oord et al. [49] For the reconstruction loss, we use cross-entropy loss with Z_t as the label. For the vector quantization objective, we randomly sample from the the distribution Z_t and acquire latent feature index Q_t as the target.

5 Experiments

In this section, we provide both quantitative and qualitative evaluation of ShapeCrafter on recursive text-conditioned shape generation task. Specifically, we focus on: (1) quantitative evaluation and comparison with AutoSDF [33] to assess the ability of ShapeCrafter to generate high-quality shapes that correspond to text input, (2) quantitative evaluation and comparison with AutoSDF [33] of the ability to handle phrase sequences of varied lengths, (3) qualitative and quantitative evaluation of recursive text-conditioned shape generation, and (4) ablations to evaluate recursive generation in preserving shape details from previous steps. For consistency, we limit our experiments to the chair category, but we show results on the table category in the supplementary document.

Dataset: All our experiments use our Text2Shape++ dataset for training (20% held out for validation). For comparisons, we use full sentences from the Text2Shape dataset and phrases from Text2Shape++ dataset for validation. **Metrics**: We use the following three metrics to evaluate text-shape correspondence and shape quality.

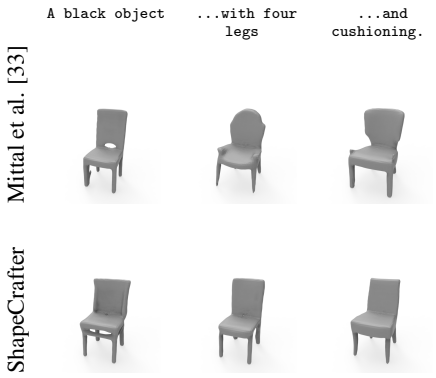


Figure 5: Qualitative comparison with AutoSDF [33]. ShapeCrafter produces sequentially more consistent shapes compared to AutoSDF.

CLIP-Similarity (CLIP-S). This metric compares the text-shape correspondence among all the methods as measured by CLIP [38]. We first rendered the generated shape into images from 20 different views. We then use the pretrained CLIP [38] to extract the text feature and images features, and calculate the similarity between the text feature and all of the shape features. We use the maximum of the similarity from the 20 view images as CLIP-Similarity. Even for the same text-shape pair, the CLIP-Similarity value can be different because of the different rendering settings. We normalize the CLIP-Similarity using two standard models, the Stanford bunny[48] and the Utah teapot[4], because we want to reduce the effect of rendering differences. We calculate the cosine similarity between rendered pictures of the Stanford bunny and the word "teapot" and use this score as a lower limit. Then, we take the max cosine similarity between pictures of the Stanford bunny and the word "bunny", and use it as the upper limit.

ShapeGlot-Confidence (SGLOT-C). The ShapeGlot-Confidence metric compares the text-shape correspondence between methods. A neural evaluator is trained as proposed in [3] to distinguish the target shape from distractors given the specified text. The trained neural evaluator achieves an 83% accuracy on this binary classification task with the Shapeglot[3] dataset. ShapeGlot-Confidence informs us which shape most closely corresponds to the text input among all shapes generated by the methods that we are comparing. We use this as a proxy for a perceptual study.

FID. We use the Frechet Inception Distance (**FID**) metric [20] to evaluate the quality of the generated shapes. For shape feature extraction, we use the voxel encoder in [55] which is trained on the ShapeNet classification task[6].

5.1 Comparisons

We evaluate and compare the performance of ShapeCrafter on text-conditioned shape generation [33]. For this experiment, we use single step generation for AutoSDF using complete prompts from the Text2Shape [7] test set (since AutoSDF is not designed for recursive generation). Our model generates recursively by taking the sentence as input, parsing it into phrases, and generating according to the parsed phrases recursively. Table 1 illustrates the performance of ShapeCrafter and AutoSDF[33] on

this task. We outperform AutoSDF on the CLIP-Similarity score and the Shapeglot-Confidence score, which means ShapeCrafter is able to generate shapes that are visually more similar to the text input. For shape quality evaluation, our model has a lower FID score, indicating that we can capture details better. This experiment also shows that the recursive generation strategy is useful, compared to the single step strategy.

In Figure 5, we also compare the performance between AutoSDF[33] and ShapeCrafter qualitatively. We visualize the shapes generated by phrases from Text2Shape++. Clearly, comparing with AutoSDF, the generated results of ShapeCrafter is more sequentially consistent.

Table 1: AutoSDF and ShapeCrafter on text-conditioned generation. Compared to AutoSDF, ShapeCrafter performs better on CLIP-S, SGLOT-C, and FID, which indicates that it provides better text-shape correspondence and shape quality.

Metric	CLIP-S ↑	SGLOT-C ↑	FID ↓
Mittal et al. [33]	48.92	0.46	18.45
ShapeCrafter (Ours)	**52.43**	**0.53**	**16.36**

5.2 Phrase Sequence Length

We are also interested in the ability of ShapeCrafter on generating text inputs of various lengths. We divide the texts inputs into three levels by the number of phrases they contain. All text inputs are categorized into three tiers: texts that contain $1-2$, $2-4$, and more than 4 phrases. Table 2 compares the performance of AutoSDF and ShapeCrafter on text inputs of various lengths. Overall, the results from these three metrics indicate that ShapeCrafter generates shapes that are more consistent with longer text inputs than AutoSDF without compromising generation quality. Specifically, for FID, the shape quality of AutoSDF decreases as the text input grows longer, while the shape quality of ShapeCrafter remains stable. For CLIP-Similarity, the performance of ShapeCrafter is comparable with AutoSDF for shorter texts, and ShapeCrafter is better than AutoSDF at generating shapes that closely agree with medium-length and long texts. For Shapeglot-Confidence, AutoSDF performs better with shorter/medium-length text inputs, but ShapeCrafter performs better with longer inputs. The result shows that recursive generation strategy outperforms single step strategy in longer text inputs.

Table 2: AutoSDF and ShapeCrafter on recursive text-conditioned shape generation. ShapeCrafter uses a recursive strategy and it performs better with longer inputs.

# Phrases	[1, 2]			(2, 4]			(4, +∞)		
	CLIP-S ↑	SGLOT-C ↑	FID ↓	CLIP-S ↑	SGLOT-C ↑	FID ↓	CLIP-S ↑	SGLOT-C ↑	FID ↓
Mittal et al.	**45.72**	**0.53**	18.13	45.27	0.42	20.33	55.77	0.43	22.47
ShapeCrafter	**45.72**	0.47	**17.40**	**53.38**	**0.58**	**16.44**	**58.18**	**0.57**	**16.89**

5.3 Recursive Text-conditioned Shape Generation

We evaluate the performance of ShapeCrafter on recursive text-conditioned shape generation task qualitatively. Fig. 6 shows the shape generated at each time step. Shapes rendered in the same color are generated from the same phrase sequences. This figure provides examples that demonstrate ShapeCrafter has the ability to (1) generate high-quality shapes *(row 1)*, (2) modify the global structure of shapes *(row 2)*, (3) change part-level attributes *(row 3-5)* including part addition, part removal, and part replacement, (4) modify the local details of the shape *(row 6)*, (5) generate from long phrase sequences *(row 7 example 1)*, (6) generate novel shapes *(row 7 example 2)*. The above properties shows ShapeCrafter performs well in text-conditioned shape generation and text-conditioned shape editing.

Table 3 shows how the probability distribution Z and generated shape changes with the number of text phrases increases. For a text phrase sequence, we calculate (a) the mean entropy(H) of the per-grid-cell discrete probability distributions $H = -\sum_{i=1}^{g^3}\sum_{k=1}^{K} z_{ik}\log(z_{ik})/g^3$, and (b) the mean Chamfer Distance(CD) among 8 randomly sampled shapes.
The table shows that the entropy and the mean Chamfer Distance decrease with more phrases. This indicates that as more phrases are added, the probabilities become narrower resulting in more deterministic shapes.

Figure 6: Qualitative results produced by ShapeCrafter. Each unique color shows results generated recursively from our model using the same seed to sample the shape distribution. Our method demonstrates consistent and gradual evolution of shape enabling us to simulate shape editing.

Table 3: The Entropy(H) of the probability distribtuion and the Chamfer Distance(CD) between the generated shapes at sentence with different number of phrases. The probabilities narrow as more phrases are added, resulting in more deterministic shapes.

# Phrases	1	$[2, 4)$	$[4, 8)$	$[8, +\infty)$
H	0.423	0.341	0.294	0.267
CD	0.0861	0.0698	0.0359	0.0261

5.4 Ablations

We design ablations to evaluate whether: (1) the conditional training strategy improves text-shape correspondence, (2) the random transformer improves shape quality; and (3) reordering transformer inputs preserves shape details. To prove the effectiveness of conditional training, we design a baseline removing the residual blocks $\Psi(\cdot)$ and only generating shape features recursively with AutoSDF [33]. Specifically, at time step t, we multiply the probability distribution C_t generated from the text input with the probability distribution generated from the random transformer at $t-1$, and we use the product as the conditional distribution of the random transformer at t. We refer to this baseline as *w/o condition*. To prove that the random transformer helps improve the quality of the generated shapes, we design a baseline *w/o transformer*, where we sample shapes from the probability distribution generated by the residual blocks $\Psi(\cdot)$. To prove the effectiveness of reordering the input sequence in the random transformer (refer to section 4.3), we design another baseline where the transformer generates features in a random sequence. We refer to this baseline as *w/o reorder*. The performance of these baselines are illustrated in Table. 4. The performance drops without these components, which proves the effectiveness of our design choices.

Table 4: Ablation studies shows the effectiveness of the conditional training, the random transformer and the reordered random transformer sequence.

Metric	CLIP-S ↑	FID ↓
w/o condition	49.53	20.22
w/o transformer	49.29	19.83
w/o reorder	42.47	16.67
Ours	**52.43**	**16.36**

6 Conclusion

In this paper, we present ShapeCrafter, a method for recursive text-conditioned 3D shape generation. Different from previous single-step methods, our focus is on generating 3D shape distributions that can be gradually evolved to capture semantics described in phrase sequences. We also presented a method to transform an existing dataset to support recursive shape generation tasks to produce a larger dataset called Text2Shape++ with 369K shape–text pairs. Results show that our method produces high-quality shapes while being able to handle long phrase sequences, support shape editing, and extrapolate to shapes unseen during training.

Limitations: Our method has several notable limitations. First, ShapeCrafter is limited to shape generation only and cannot handle appearance attributes common in language descriptions. Limitations of the Text2Shape dataset extend to our method—we only support two shape categories: tables and chairs. Empirically, we observe that our method demonstrates elements of "part assembly", but cannot handle large-scale deformations. The proposed recursive shape generation method is not reversible. For example, given an original shape with inputs "with armrest" and "without armrest" applied sequentially, the generated shape may not be identical to the original shape. Another noticeable problem in the text-conditioned shape editing community is that there lacks a common metric to evaluate the accuracy of shape editing. We believe that we have taken a step in the direction towards addressing the above challenges. From a **societal impact** perspective, more care is needed to ensure that our dataset represents diverse shape instances to avoid bias.

Acknowledgements This research was supported by AFOSR grant FA9550-21-1-0214, NSF CNS-2038897, and the Google Research Scholar Program. Daniel Ritchie is an advisor to Geopipe and owns equity in the company. Geopipe is a start-up that is developing 3D technology to build immersive virtual copies of the real world with applications in various fields, including games and architecture.

References

[1] Speech-to-text: Automatic speech recognition | google cloud.

[2] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017.

[3] Panos Achlioptas, Judy Fan, X.D. Robert Hawkins, D. Noah Goodman, and J. Leonidas Guibas. ShapeGlot: Learning language for shape differentiation. *CoRR*, abs/1905.02925, 2019.

[4] James F Blinn and Martin E Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, 1976.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[7] Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. Text2shape: Generating shapes from natural language by learning joint embeddings. In *Asian conference on computer vision*, pages 100–116. Springer, 2018.

[8] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[9] Noam Chomsky. Minimal recursion: exploring the prospects. In *Recursion: Complexity in cognition*, pages 1–15. Springer, 2014.

[10] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[11] Michael C Corballis. From hand to mouth. In *From Hand to Mouth*. Princeton University Press, 2020.

[12] Angela Dai and Matthias Nießner. Scan2mesh: From unstructured range scans to 3d meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5574–5583, 2019.

[13] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Lrec*, volume 6, pages 449–454, 2006.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[15] Alaaeldin El-Nouby, Shikhar Sharma, Hannes Schulz, Devon Hjelm, Layla El Asri, Samira Ebrahimi Kahou, Yoshua Bengio, and Graham W Taylor. Keep drawing it: Iterative language-based image generation and editing. *arXiv preprint arXiv:1811.09845*, 2, 2018.

[16] Stanislav Frolov, Tobias Hinz, Federico Raue, Jörn Hees, and Andreas Dengel. Adversarial text-to-image synthesis: A review. *Neural Networks*, 144:187–209, 2021.

[17] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9785–9795, 2019.

[18] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[19] Xiaodong He and Li Deng. Deep learning for image-to-text generation: A technical overview. *IEEE Signal Processing Magazine*, 34(6):109–116, 2017.

[20] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[21] Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[22] Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. *arXiv preprint arXiv:2112.01455*, 2021.

[23] Danilo Jimenez Rezende, SM Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. *Advances in neural information processing systems*, 29, 2016.

[24] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[25] Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[26] Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, and Philip H. S. Torr. Controllable text-to-image generation. *arXiv preprint arXiv:1909.07083*, 2019.

[27] Chun-Liang Li, Manzil Zaheer, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov. Point cloud gan. *arXiv preprint arXiv:1810.05795*, 2018.

[28] Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5162–5170, 2015.

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[30] Zhengzhe Liu, Yi Wang, Xiaojuan Qi, and Chi-Wing Fu. Towards implicit text-guided 3d shape generation. *arXiv preprint arXiv:2203.14622*, 2022.

[31] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[32] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[33] Paritosh Mittal, Yen-Chi Cheng, Maneesh Singh, and Shubham Tulsiani. Autosdf: Shape priors for 3d completion, reconstruction and generation. *arXiv preprint arXiv:2203.09516*, 2022.

[34] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[35] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *European Conference on Computer Vision*, pages 523–540. Springer, 2020.

[36] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[37] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[38] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[39] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[40] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[41] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[42] Aditya Sanghi, Hang Chu, Joseph G Lambourne, Ye Wang, Chin-Yi Cheng, and Marco Fumero. Clip-forge: Towards zero-shot text-to-shape generation. *arXiv preprint arXiv:2110.02624*, 2021.

[43] Matan Shoef, Sharon Fogel, and Daniel Cohen-Or. Pointwise: An unsupervised point-wise feature learning network. *arXiv preprint arXiv:1901.04544*, 2019.

[44] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems*, 32, 2019.

[45] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11358–11367, 2021.

[46] Ming Tao, Hao Tang, Songsong Wu, Nicu Sebe, Xiao-Yuan Jing, Fei Wu, and Bingkun Bao. Df-gan: Deep fusion generative adversarial networks for text-to-image synthesis. *arXiv preprint arXiv:2008.05865*, 2020.

[47] Shubham Tulsiani and Abhinav Gupta. Pixeltransformer: Sample conditioned signal generation. *arXiv preprint arXiv:2103.15813*, 2021.

[48] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, 1994.

[49] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[50] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[52] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. *arXiv preprint arXiv:2112.05139*, 2021.

[53] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018.

[54] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[55] Udaranga Wickramasinghe, Edoardo Remelli, Graham Knott, and Pascal Fua. Voxel2mesh: 3d mesh model generation from volumetric data. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 299–308. Springer, 2020.

[56] Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. Cnns on surfaces using rotation-equivariant features. *ACM Transactions on Graphics (TOG)*, 39(4):92–1, 2020.

[57] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1316–1324, 2018.

[58] Xingguang Yan, Liqiang Lin, Niloy J Mitra, Dani Lischinski, Danny Cohen-Or, and Hui Huang. Shapeformer: Transformer-based shape completion via sparse representation. *arXiv preprint arXiv:2201.10326*, 2022.

[59] Hui Ye, Xiulong Yang, Martin Takac, Rajshekhar Sunderraman, and Shihao Ji. Improving text-to-image synthesis using contrastive learning. *arXiv preprint arXiv:2107.02423*, 2021.

[60] Han Zhang, Jing Yu Koh, Jason Baldridge, Honglak Lee, and Yinfei Yang. Cross-modal contrastive learning for text-to-image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 833–842, 2021.

[61] Minfeng Zhu, Pingbo Pan, Wei Chen, and Yi Yang. Dm-gan: Dynamic memory generative adversarial networks for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5802–5810, 2019.

Supplementary Document

In this supplementary document, we provide more details about:

1. Real-time recursive audio-based 3D shape generation.
2. Additional experiment results comparing with other text-conditioned 3D shape generation methods.
3. Visualization of probability distribution changes as text phrase are added.
4. Failure cases of ShapeCrafter.
5. Visualization of Text2Shape++ examples.
6. Objective Functions.

7 Real-Time Recursive Audio-based 3D Shape Generation

We provide a real-time recursive audio-based 3D shape generation demonstration. Please refer to the video attached to the supplementary material. We use the Google Speech Recognition[1] library to convert audio input to text, and then feed the text to ShapeCrafter. At each step, the user describes the target shape, and ShapeCrafter will generate multiple results accordingly. To generate diverse results and accelerate the inference step, in the video we use one random seed but repeat the same text at batch dimension instead of using different seeds. The generated shapes are gradually evolving as more descriptions are given. We show some screenshots below – we highly encourage readers to watch the supplementary video.

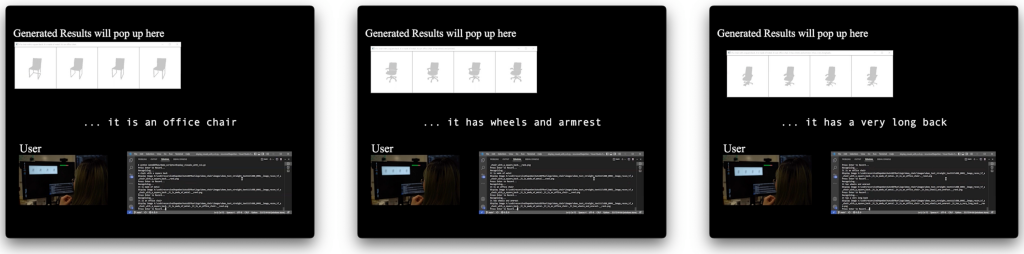


Figure 7: Screenshots of our real-time demo (please see supplementary video).

8 Additional Experimental Results

We compare our method with other text-condtioned 3D shape generation methods, including [7], [42], and [30]. Table 5 compares the results on the original Text2Shape dataset. We also compare with [30] on generating 3D shapes from varied-length text inputs in Table 6. Our method performs well on all of the metrics, which means that ShapeCrafter is comparable with the state-of-the-art in terms of the shape quality and the text-shape correspondence. ShapeCrafter has slight advantage on *CLIP-S* and *Shapeglot-C* with longer texts, which shows the effectiveness of recursive generation.

Table 5: Comparison with other methods on text conditioned generation.

Metric	CLIP-S ↑	Shapeglot-C ↑	FID ↓
Chen et al. [7]	16.29	0.14	20.21
Sanghi et al. [42]	26.34	0.25	21.50
Liu et al. [30]	38.88	**0.50**	16.91
ShapeCrafter	**52.43**	**0.50**	**16.36**

We also provide more qualitative results on the table category in Fig. 8. It shows that ShapeCrafter can generalize to categories other than chairs.

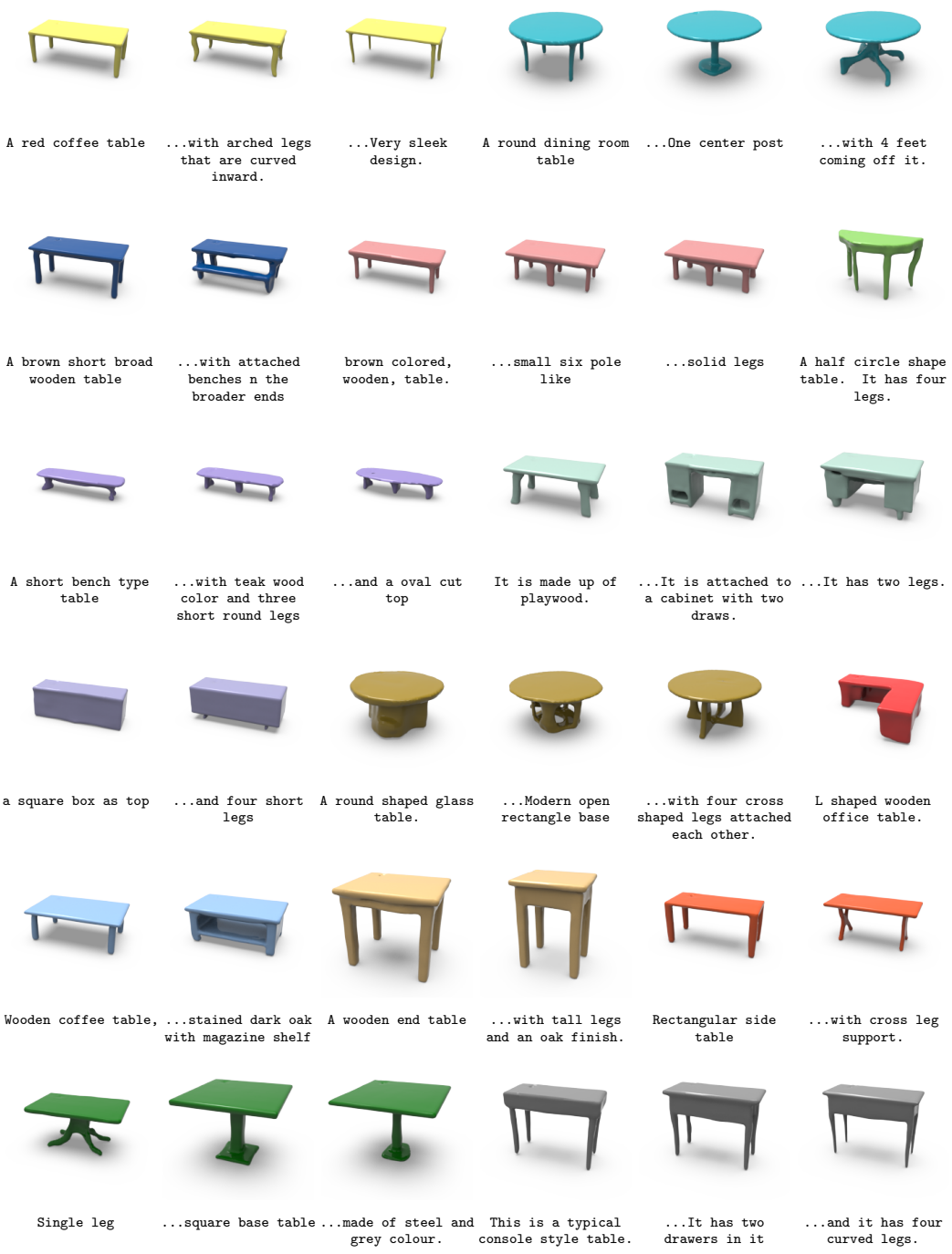


Figure 8: Qualitative results produced by ShapeCrafter. Each unique color shows results generated recursively from our model using the same seed to sample the shape distribution. Our method demonstrates consistent and gradual evolution of shape enabling us to simulate shape editing.

9 Visualization of the Probability Distribution Shift

In this section, we evaluate and visualize how the probability distribution Z changes with text input. We calculate the probability distribution difference of shape features for each grid cell at two

Table 6: Comparison with state-of-the-art on recursive text-conditioned Generation.

# Phrases	[1, 2]			(2, 4]			(4, +∞)		
	CLIP-S ↑	Shapeglot-C ↑	FID ↓	CLIP-S ↑	Shapeglot-C ↑	FID ↓	CLIP-S ↑	Shapeglot-C ↑	FID ↓
Liu et al.	27.20	**0.61**	**17.30**	42.32	0.46	17.80	42.84	0.48	**16.88**
ShapeCrafter	**45.72**	0.39	17.40	**53.38**	**0.54**	**16.44**	**58.18**	**0.52**	16.89

consecutive time step with: diff $= \max(z_{t,k} - z_{t-1,k}) \in [0, 1]$, where $k \in 1, 2, ..., K$, and K is the number of codes in the codebook. At two consecutive time steps, we report the ratio of grid cells whose probability distribution difference is smaller than a threshold τ among all 8^3 grid cells. In the Table. 7, we evaluate the local geometry change with the *percentage of distribution difference metric*. We compare our method with AutoSDF.

Table 7: The percentage of distribution difference under threshold τ.

τ	1e-10	1e-9	1e-8	1e-7	1e-6
Liu et al. [30]	1.70	4.31	9.65	17.1	25.08
ShapeCrafter	6.98	11.96	18.03	24.17	29.56

The table shows that ShapeCrafter has lower percentage change of grid cell probability from step to step than AutoSDF[33], which shows that recursive generation changes localized regions. We acknowledge that this metric cannot definitely prove that the changed region semantically corresponds to the change in the input text; this is very hard to evaluated with a single number.

We visualize how the probability distribution Z is shifted by text inputs in Figure 9. The four rows illustrate: the text inputs, the generated shapes, the SDF values in the resolution of 64^3 projected to a plane, and the difference of the probability distribution in the resolution of 8^3 projected to a plane, respectively. We calculate the probability distribution difference of grid at two consecutive time step with:

$$\text{diff} = \frac{|z_{t,k} - z_{t-1,k}|}{z_{t-1,k}} \tag{7}$$

where $z_{t,k}$ stands for the probability of the index of the discrete latent feature being k at time t. Clearly, the probability distributions of the discrete latent feature are locally excited by text inputs. For instance, with the addition of text input "while the hand rests are cylindrical in shape", the probability significantly changes in the portion of the grid where the armrests would appear. Since most chairs with armrests have a back that leans backwards, the back portion of the grid also experiences a significant change in probability.

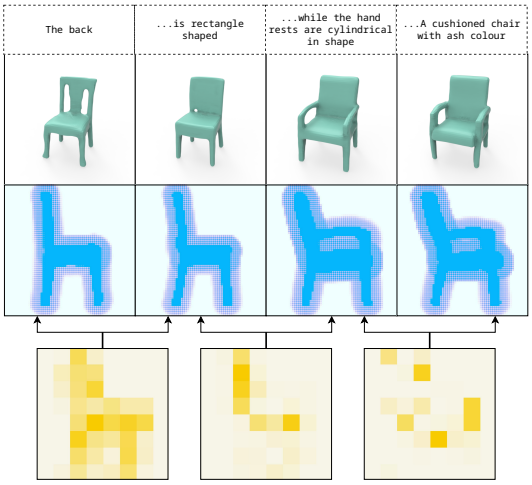
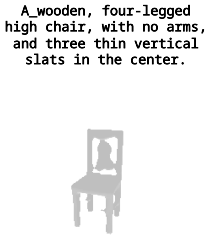


Figure 9: Visualization of text inputs, generated shapes, SDF values and probability distribution shifts. Darker yellow corresponds to greater difference.

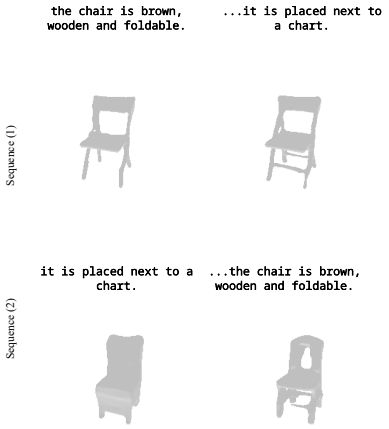
10 Failure Cases

We analyze the typical failure cases of ShapeCrafter in this section. Fig. 10 demonstrates the failure cases.

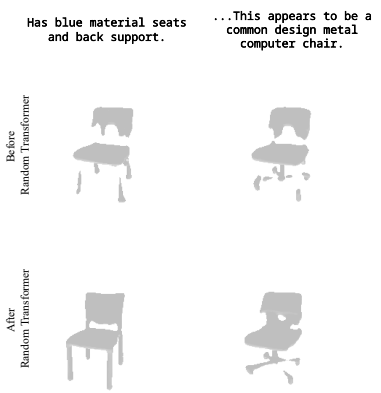
Figure 10a shows a failure case caused by shape representation. The generated shape semantically corresponds to the text input, but it fails to generate "three thin vertical slats". Although the vector-quantized grid-based neural implicit representation[33] can represent shape efficiently and effectively,



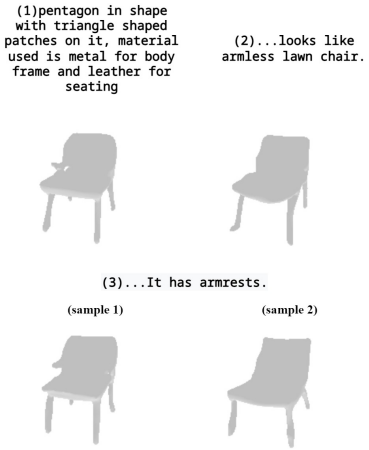
(a) Failure case resulted by shape representation.



(b) Failure case resulted by global editing.



(c) Failure case resulted by random transformer.



(d) Failure case resulted by conflict attributes.

Figure 10: Failure cases generated by ShapeCrafter.

it fails to represent shape with fine-grained details. A possible solution is to build a hierarchical vector-quantized latent space that preserves more details.

Figure 10b shows a failure case caused by global editing, which is a limitation of our method. In sequence (1), the model recursively generates a "foldable chair" from the phrase sequence. However, if we switch the order of the phrases, it fails to generate a "foldable chair". This results from the fact that a "foldable chair" globally differs from the shape generated in the previous step, so ShapeCrafter fails to edit it in a reasonable way. A possible solution is that we can find the head-noun in the sentence using dependency parsing[13] and generate the first shape according to the head noun in the sentence.

Figure 10c shows a failure case caused by the random transformer[33]. In this figure, the model is modifying the legs of the chair. If we reconstruct the shape from the grid features output by the Residual Blocks, the chair backs look similar. However, if we reconstruct the shape from the grid features output by the random transformer, the chair backs look different. A possible solution is to learn a confidence map, where a higher confidence score will result in a higher likelihood of the grid

being edited at that time step. The transformer only auto-regresses on the grid features where the confidence score is high.

Figure 10d shows a failure case caused by conflict attributes. In this figure, two consecutive text phrases first describe the chair as "armless", and then describe that it "has armrests", which is a pair of conflict attributes. Sampled with different seeds, ShapeCrafter sometimes generates chairs with armrests and sometimes generates armless chairs. This could be due to the fact that our model represents shapes as probability distributions.

11 Objective Functions

We provide the objective functions for training the (1) P-VQ-VAE model for shape representation; (2) Auto-regressive model for shape generation; (3) $\mathbf{BERT}_{BASE}$ model for text feature extraction; (4) Text feature projection model Φ; and (5) Residuals blocks Ψ to extract the final distribution.

11.1 P-VQ-VAE Model

To train the P-VQ-VAE model, we use the reconstruction loss, the VQ loss and the commitment loss. For the shape X, the loss is formulated as:

$$L = \frac{1}{T}\sum_{i=1}^{T}(BCE(D_\phi(VQ(E_\phi(x_i))), o_i) + \alpha \left\| sg[VQ(E_\phi(x_i))] - e_i' \right\|_2^2 + \beta \left\| VQ(E_\phi(x_i)) - sg[z_i] \right\|_2^2), \tag{8}$$

where BCE is the binary cross entropy loss, $sg[\cdot]$ is the stop gradient operation, x_i is the sampled coordinate point, o_i is the target occupancy of the sampled coordinate point, e_i' is the nearest latent feature in the codebook, T is the total points sampled in the space, and α and β are weighting factors.

11.2 Text Feature Extraction Models

The text feature extraction models include $\mathbf{BERT}_{BASE}$, Projection model $\Phi(\cdot)$, and Residuals blocks $\Psi(\cdot)$, which are trained together with the following loss function:

$$L = CE(\Psi(\Phi(\mathbf{BERT}_{BASE}(I))), Q^{set}), \tag{9}$$

where CE is the cross entropy loss, I is the text input, and Q^{set} is the voxel grids of the index code.

11.3 Auto-regressive Model

To train the auto-regressive model $f(\cdot)$, we have:

$$L = CE(f(Z), Q^{set}), \tag{10}$$

where CE is the cross entropy loss, Z is the voxel grids of probability distribution and Q^{set} is the voxel grids of the index code.