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

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. Network architecture and loss functions.

1 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.

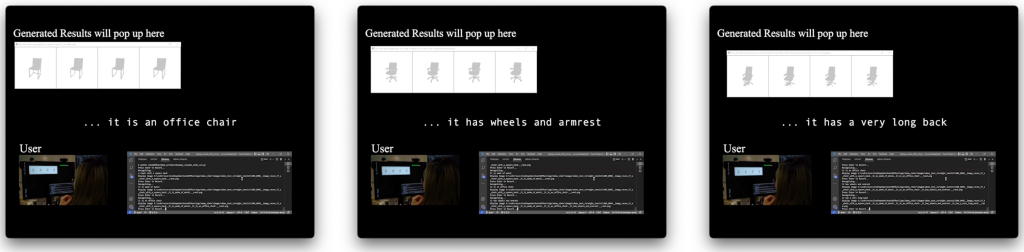


Figure 1: Screenshots of our real-time demo (please see supplementary video).

2 Additional Experimental Results

We compare our method with other text-condtioned 3D shape generation methods, including [2], [6], and [4]. Table 1 compares the results on the original Text2Shape dataset. We also compare with [4] on generating 3D shapes from varied-length text inputs in Table 2. 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.

We also provide more qualitative results on the table category in Fig. 2. It shows that ShapeCrafter can generalize to categories other than chairs.

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

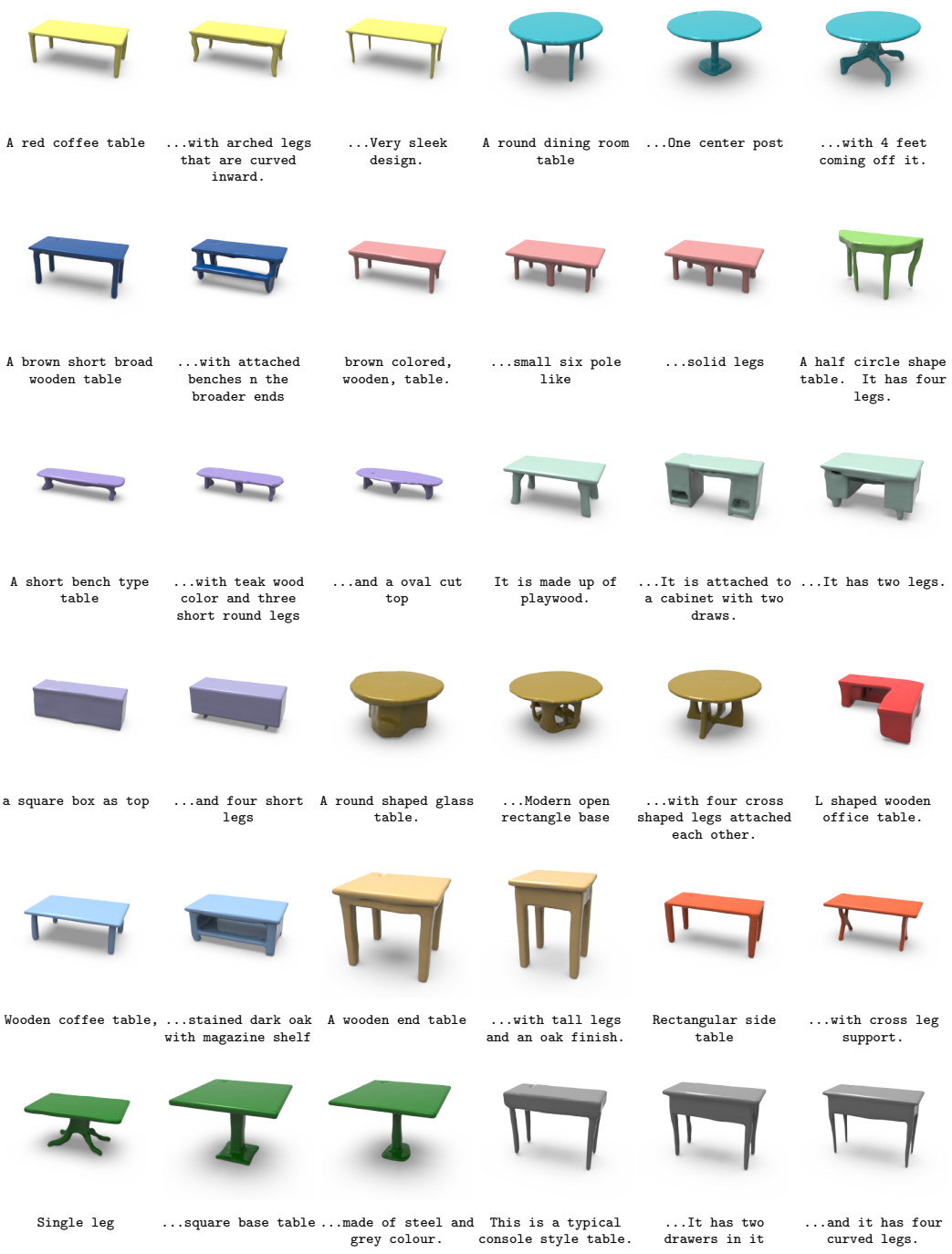


Figure 2: Qualitative results produced by ShapeCrafter. Each unique color shows results generated recursively from our model using the same seed to sample the shape distribution. Our method demonstrates consistent and gradual evolution of shape enabling us to simulate shape editing.

3 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 1: Comparison with other methods on text conditioned generation.

Metric	CLIP-S ↑	Shapeglot-C ↑	FID ↓
Chen et al. [2]	16.29	0.14	20.21
Sanghi et al. [6]	26.34	0.25	21.50
Liu et al. [4]	38.88	**0.50**	16.91
ShapeCrafter	**52.43**	**0.50**	**16.36**

Table 2: 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. 3, we evaluate the local geometry change with the *percentage of distribution difference metric*. We compare our method with AutoSDF.

Table 3: The percentage of distribution difference under threshold τ.

τ	1e-10	1e-9	1e-8	1e-7	1e-6
Liu et al. [4]	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[5], 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 3. 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:

$$\text{diff} = \frac{|z_{t,k} - z_{t-1,k}|}{z_{t-1,k}} \quad (1)$$

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.

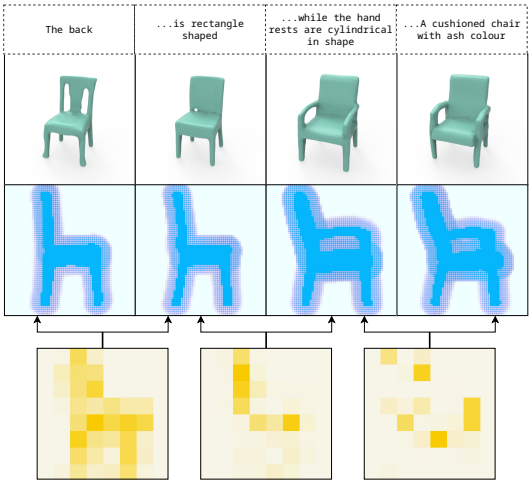
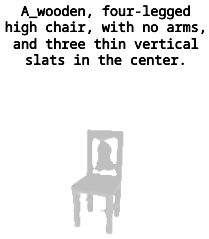
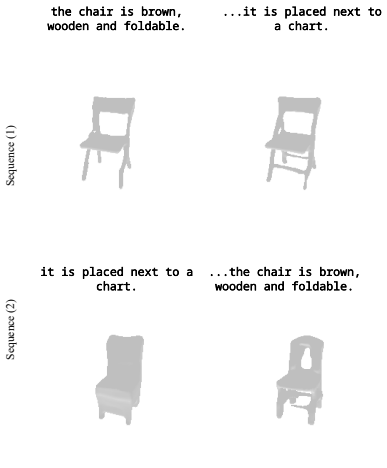


Figure 3: Visualization of text inputs, generated shapes, SDF values and probability distribution shifts. Darker yellow corresponds to greater difference.

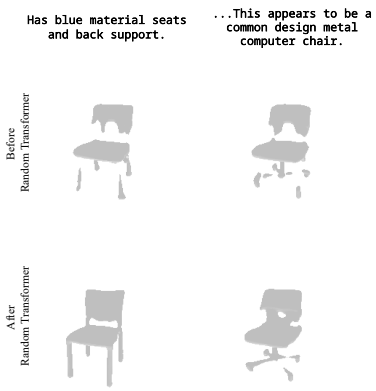
4 Failure Cases



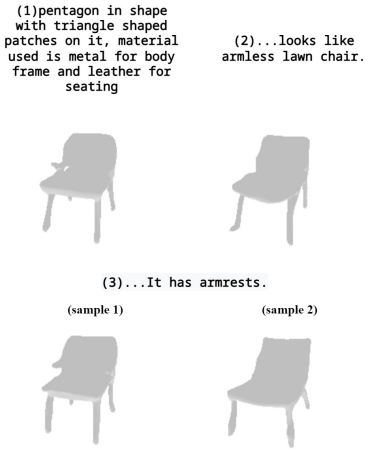
(a) Failure case resulted by shape representation.



(b) Failure case resulted by global editing.



(c) Failure case resulted by random transformer.



(d) Failure case resulted by conflict attributes.

Figure 4: Failure cases generated by ShapeCrafter.

We analyze the typical failure cases of ShapeCrafter in this section. Fig. 4 demonstrates the failure cases.

Figure 4a 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[5] can represent shape efficiently and effectively, 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 4b 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[3] and generate the first shape according to the head noun in the sentence.

Figure 4c shows a failure case caused by the random transformer[5]. 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 4d 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.

5 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.

5.1 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:

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

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.

5.2 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:

$$L = CE(\Psi(\Phi(\mathbf{BERT}_{BASE}(I))), Q^{set}), \tag{3}$$

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

5.3 Auto-regressive Model

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

$$L = CE(f(Z), Q^{set}), \tag{4}$$

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.

References

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

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.