NEURAL SHAPE COMPILER: A UNIFIED FRAMEWORK FOR TRANSFORMING BETWEEN TEXT, POINT CLOUD, AND PROGRAM

Anonymous authors
Paper under double-blind review

ABSTRACT

3D shapes have complementary abstractions from low-level geometry to part-based hierarchies to languages, which convey different levels of information. This paper presents a unified framework to translate between pairs of shape abstractions: *Text* $\Longleftrightarrow$ *Point Cloud* $\Longleftrightarrow$ *Program*. We propose ***Neural Shape Compiler*** to model the abstraction transformation as a conditional generation process. It converts 3D shapes of three abstract types into unified discrete shape code, transforms each shape code into code of other abstract types through the proposed *ShapeCode Transformer*, and decodes them to output the target shape abstraction. Point Cloud code is obtained in a class-agnostic way by the proposed *Point*VQVAE. On Text2Shape, ShapeGlot, ABO, Genre, and Program Synthetic datasets, Neural Shape Compiler shows strengths in *Text* $\Longrightarrow$ *Point Cloud*, *Point Cloud* $\Longrightarrow$ *Text*, *Point Cloud* $\Longrightarrow$ *Program*, and Point Cloud Completion tasks. Additionally, Neural Shape Compiler benefits from jointly training on all heterogeneous data and tasks.

1 INTRODUCTION

Humans understand 3D shapes from different perspectives: we perceive geometries, understand their composing parts and regularities, and describe them with natural language. Similarly, vision researchers designed different abstractions for 3D shapes, including (1) low-level geometric structures that plot detailed geometry such as point clouds (Gruen & Akca, 2005); (2) structure-aware representations that can tell shape parts and their relations, like shape programs (Tian et al., 2019); (3) natural language to describe compositionality and functionality (Çağdaş, 1996; Chang et al., 2014). Different shape abstractions convey complementary information and encode different characteristics allowing researchers to design specialized models for various tasks (Mitra & Nguyen, 2003; Chaudhuri et al., 2020). This paper studied how to translate between *Text* $\Longleftrightarrow$ *Point Cloud* $\Longleftrightarrow$ *Program* for providing multi-level information about shape (Figure 1, 8). This transformation mechanism may also facilitate shared progress among tasks accomplished by specialized models, such as translating successes in shape completion into advances in shape structure understanding (Figure 7).

Modern program compilers (Lattner & Adve, 2004; Zakai, 2011) turn source codes into intermediate representations (*IRs*), transform *IRs*, and decode them into other types of high-level programming languages. We take inspiration here and propose a unified architecture, ***Neural Shape Compiler***, to perform compilation between the three shape abstractions with neural networks (Rumelhart et al., 1986). Neural Shape Compiler converts all shape abstractions into unified discrete shape codes (i.e., IR) via their respective encoders. Then, *ShapeCode Transformer* transforms shape codes of one type to another in an autoregressive and probabilistic manner (Ramesh et al., 2021), where the probabilistic way helps us overcome the issue of ambiguity that different shape geometries can correspond to very similar program or text. Finally, the compiled shape code is decoded to the target shape abstraction through the corresponding decoder[1]. Figure 1 shows the entire process and three example compilations.

To turn point clouds with continuous coordinates into discrete shape code, we propose ***Point*VQVAE** inspired by (Van Den Oord et al., 2017). Unlike traditional point cloud process models (Qi et al., 2017a), our encoder has restricted receptive fields (Coates & Ng, 2011; Luo et al., 2019) for learning

[1] *Text* $\Longleftrightarrow$ *Program* is achieved in two-step generation *Text* $\Longleftrightarrow$ *Point Cloud* $\Longleftrightarrow$ *Program*. *Program* $\Rightarrow$ *Point Cloud* is deterministic via executing the programs

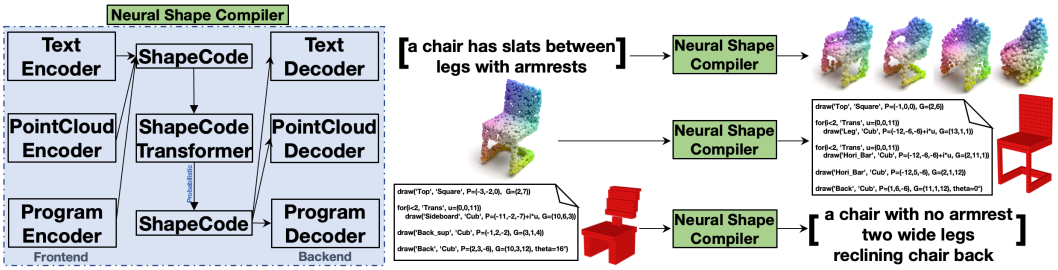


Figure 1: Overview, and three compilations conducted by Neural Shape Compiler, including *Text* $\Rightarrow$ *Point Cloud*, *Point Cloud* $\Rightarrow$ *Program*, and *Program* $\Rightarrow$ *Text*[1]. Red shapes are the rendered results by executing the corresponding program. ShapeCode Transformer is shared across different compilations. Neural Shape Compiler is not limited to performing the compilations shown above, and benefits from joint training with all heterogeneous data and tasks.

shape part embeddings. Those part embeddings let the codebook (Van Den Oord et al., 2017) encode the parts of the input 3D shape rather than the entire shape. The decoder then combines all the 3D part codes to reconstruct whole point clouds via a residual-like structure (He et al., 2016) in a permutation-equivalent manner.

This paper focuses on modeling 3D shape structures. Our text data contains detailed structural descriptions and is devoid of color and texture, where we adjusted and annotated data from Text2Shape (Chen et al., 2018), ShapeGlot (Achlioptas et al., 2019), and ABO (Collins et al., 2021) datasets, resulting in 107,371 (*Point Cloud*, *Structure-Related Text*) pairs. We synthesized 120,000 (*Point Cloud*, *Program*) pairs (Tian et al., 2019) for understanding shape parts and regularities with programs. Our experiments demonstrate that Neural Shape Compiler achieves strong performance in *Text* $\Longrightarrow$ *Point Cloud*, *Point Cloud* $\Longrightarrow$ *Text*, and *Point Cloud* $\Longrightarrow$ *Program* tasks, and can be further boosted by joint training with all heterogeneous data and tasks. In addition, Neural Shape Compiler shows positive signals that can generate point clouds corresponding to input text with geometric details, text that can describe the structure of the input point clouds, and programs that can tell compositions. Besides, our experiments suggest that CLIP (Radford et al., 2021) may not be suitable for generating 3D shapes with structural details, which is used in some recent *Text* $\Longrightarrow$ *3D* methods (Jain et al., 2021; Sanghi et al., 2022; Poole et al., 2022).

Neural Shape Compiler is extensible. We show a case where our framework is extended to perform Point Cloud completion by transforming partial Point Cloud code into complete Point Cloud code. Limitations and future work are discussed in Appendix C. Our code, pre-trained models, and datasets will be released to facilitate future research in 3D multimodal learning.

2 Related Work

Compiler: Our proposed framework is inspired by the concept of computer program compiler (Calingaert, 1979; Appel, 2004; Aho et al., 2007; Aho & Ullman, 2022). Classical compilers translate high-level program language (e.g., Pascal, C) into executable machine code with the help of assemblers and linkers (Wirth et al., 1996; Appel, 2004), while Neural Shape Compiler does the bidirectional transformation between different shape abstractions with the help of ShapeCode Transformer. Compared to classical compilers, our framework is more similar to modern program compilers (e.g., LLVM (Lattner & Adve, 2004) and Emscripten (Zakai, 2011)) where the target code is not limited to machine code and can be high-level programming languages. Our framework shares similar architectures with LLVM, as shown in the left of Figure 1: LLVM uses front-ends (encoders) to turn the corresponding source codes into intermediate representations (shape codes) and decode them via back-ends (decoders). A major difference between our framework and modern program compilers is that our IR transformation process is probabilistic, whereas the process in a program compiler is a deterministic process with potential performance optimizations.

Multimodal Learning: With web-scale image-text data, researchers achieved remarkable progress in multi-modal learning of 2D-text (Li et al., 2019; Ramesh et al., 2021; Radford et al., 2021; Patashnik et al., 2021; Alayrac et al., 2022; Ramesh et al., 2022). However, due to the lack of large-scale 3D-text pairs and baseline systems (He et al., 2017; Jaegle et al., 2021), there are slow progress in 3D-text

modeling. Some recent works tried to leverage the progress in 2D-text multimodal learning (e.g., CLIP, Imagen (Saharia et al., 2022) (Radford et al., 2021)) for 3D-text modeling (Jain et al., 2021; Zhang et al., 2021; Sanghi et al., 2022; Liu et al., 2022a; Poole et al., 2022). However, our experiments suggest CLIP may not be suitable for generating 3D shapes with structural details (Appendix B). Compared to them, this work studies the connections between 3D and text directly (Chen et al., 2018; Fu et al., 2022), as there are significant compositional connections between shape parts and words. Our work is closer to Text-to-Voxel works (Chen et al., 2018; Liu et al., 2022b), while none of them can generate desirable shapes corresponding to text prompt with levels of geometric details. Mittal et al. (2022) concurrently studied language-guided shape generation with learned autoregressive shape prior. This paper focuses on modeling 3D-text in structure aspects, which is less related to the works (Michel et al., 2022; Gao et al., 2022a), which studied stylizing 3D shapes with texts.

Generative Model: *Point*VQVAE is inspired from VQVAE (Van Den Oord et al., 2017) which studied variational auto-encoder (Kingma & Welling, 2013) with discrete latent variables (i.e., discrete codes) (Salakhutdinov & Larochelle, 2010). We leverage its most basic concept, vector quantization (Theis et al., 2017; Agustsson et al., 2017; Oord et al., 2016), and use the discrete codes as our *IRs* in Neural Shape Compiler. We did not exhaust the complex variants or techniques for generality, such as Gumbel-softmax (Jang et al., 2016; Ramesh et al., 2021), the exponential moving average in codebook updating, and multi-scale structures (Razavi et al., 2019; Vahdat & Kautz, 2020). Prior works studied learning 3D shape probabilistic spaces with GANs for shape synthesis (Wu et al., 2016; Nguyen-Phuoc et al., 2019); (Achlioptas et al., 2018; Yang et al., 2019; Cai et al., 2020) learn the continuous distribution of point clouds and sample thereon to generate shapes; (Mittal et al., 2022; Cheng et al., 2022; Yan et al., 2022) concurrently developed ways to quantize point clouds and voxels; diffusion models are also exploited in 3D shapes (Luo & Hu, 2021; Zhou et al., 2021).

3D Shape Understanding: Our approach is related to research on 3D shape understanding, including geometry processing and structural relationship discovery. Recent learning systems for shape processing usually perform over some low-level geometry representations, such as point clouds (Bronstein & Kokkinos, 2010; Qi et al., 2017b; Liu et al., 2019; Luo et al., 2020), volumetric grids (Wu et al., 2015; Maturana & Scherer, 2015; Wu et al., 2016; Wang et al., 2019), multi-view images (Bai et al., 2016; Feng et al., 2018; Han et al., 2019), meshes (Groueix et al., 1802; Lahav & Tal, 2020; Hu et al., 2022), and implicit functions (Mescheder et al., 2019; Park et al., 2019; Genova et al., 2020). Some represent shapes in a more structured way, including hierarchies, trees, and graphs (Wang et al., 2011; Li et al., 2017; Sharma et al., 2018; Mo et al., 2019). Neural Shape Compiler adopts point cloud as one of its shape abstractions because point cloud can describe shape structures and can be easily acquired by real sensors. Regarding the types of shape regularities (Mitra et al., 2006; Pauly et al., 2008; Mitra et al., 2007; Wang et al., 2011; Xu et al., 2012), this paper mainly considers extrinsic part-level symmetries, including transformational, reflectional, and rotational symmetries, which relate to part pairs and multiple parts. One of the recent advances in relationship discovery is the shape program (Tian et al., 2019; Jones et al., 2020; 2021; Cascaval et al., 2022), which designed a shape domain-specific language to encode shape relationships implicitly in programs.

3 METHOD

This paper aims to build a unified framework that can transform different shape abstractions and benefit from joint multimodal learning. The framework, shown in Figure 1, has three components: encoders, ShapeCode Transformer, and decoders. Each type of input shape (*Point Cloud*, *Text*, or *Program*) is converted to unified discrete shape code $([c_1, ..., c_n], \{c_i\} \in \mathbb{Z}+)$ through the corresponding encoder. ShapeCode Transformer then transforms each shape code into the code that can be decoded into target shape abstraction via the corresponding decoder. Each type of encoder and decoder are designated to consume the specific type of shape, while ShapeCode Transformer communicates with all encoders and decoders via discrete code and handles all types of transformations.

3.1 ENCODERS & DECODERS

To turn point clouds $P = \{p_1, \cdots, p_m\}$ $(p_i = (x_i, y_i, z_i))$ into codes $\mathcal{C}^{point} = ([c^p_1, c^p_2, ..., c^p_{N_{point}}], \{c^p_i\} \in \mathbb{Z}+)$ while accommodating complex variations of shape structures, we take inspirations from VQVAE (Van Den Oord et al., 2017) and propose *Point*VQVAE to conduct this task in a class-agnostic way. The structure of *Point*VQVAE is shown in Figure 2, where we designed a new encoder and decoder to let it successfully reconstruct point clouds from codes.

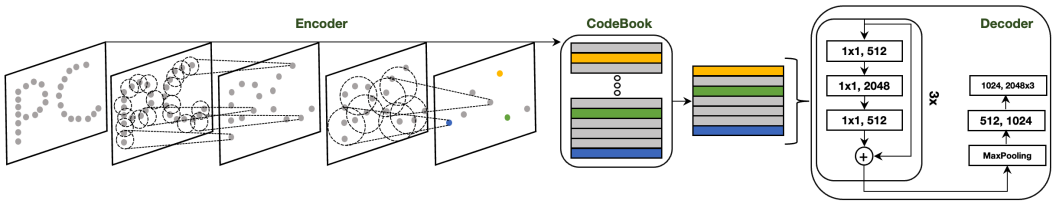


Figure 2: *Point*VQVAE consists of three parts: (1) a hierarchical encoder that has restricted receptive fields and outputs multiple part embeddings; (2) a shared codebook for vector quantization (Van Den Oord et al., 2017) (e.g., looking up the closest yellow embedding in the codebook for the yellow output of the encoder); (3) a decoder with multiple residual building blocks consisting of 1x1 convolutional layers, a max-pooling layer, and MLP layers at the end to output 2,048 points.

For the encoder, the key point is to constrain its receptive fields (Luo et al., 2016) of the last layer neurons to very local regions (Luo et al., 2019), and thus, let the codebook that follows learn how to encode parts of 3D shapes instead of entire 3D shapes (Luo et al., 2020). To achieve this, we propose a shallow hierarchical encoder, implemented as a PointNet++-like structure (Qi et al., 2017b) without losing generality. In each turn T_i, we first do farthest point sampling over input point clouds to sample a set of points noted as center points. Around each center point, we draw a ball with a radius R_{T_i} to gather information from all points within the query ball. After this, the input point clouds will be downsampled into smaller point clouds with updated embeddings on each point. Since the radius R_{T_i} of our query ball is usually smaller than the scale of the whole point clouds, the receptive field of points in downsampled point clouds can be gradually increased as we repeatedly the downsampling process (i.e., increase $\#T$). In our experiments, we set the number of rounds $\#T = 2$ and R_{T_i} to be small values ($R_{T_1} = 0.1, R_{T_2} = 0.4$) for letting the final neurons (e.g., colorful points of the encoder in Figure 2) only receive local part information regarding the input shapes. Without this design, *Point*VQVAE will fail to auto-encode various shapes (Appendix F).

For each embedding in the last layer of our encoder, we look up its closest embedding in our codebook. We concatenate all the obtained embedding together as the input to our decoder. The concatenated embedding encodes the full information of the input shape by collecting the information encoded by the different local parts of the input shape. Compared to common point clouds auto-encoders (Qi et al., 2017b), we discard the position / coordinate information of the final points of the encoder. Not using positional information may degrade auto-encoding performance but allow us to translate between codes successfully where we do not have any positional information within the source codes. To compensate for this point, we designed a residual-network-like block (He et al., 2016) and made our decoder with great depth to exploit the information from the concatenate embedding fully. The detailed structure of our decoder is in Figure 2. A max-pool layer is added to the decoder's output to ensure the final embedding is invariant to the concatenation order of the embedding (Qi et al., 2017a). In our experiments, we adopt Chamfer distance (Fan et al., 2017) and Earth Mover's distance (Rubner et al., 2000) as our reconstruction loss and the straight-through gradient estimator (Bengio et al., 2013) for backpropagating gradients to the encoder by copying the gradients from the inputs of the decoder to the encoder output. We followed the codebook objectives used in (Van Den Oord et al., 2017) to update our codebook by moving our embedding vectors closer to the encoder outputs and preventing our embedding from growing too big. Our codebook embedding space is $[512, 512]$.

We adopt the simplest way to encode and decode texts and programs for generality. We use the standard method of processing texts that leverage BPE (Sennrich et al., 2015) to encode and decode lowercase descriptions. According to the domain-specific language of shape program (Tian et al., 2019), it has limited discrete types of statements $\{A, B, C, \cdots\}$ and ranges of parameters for each statement $\{a_1, a_2, \cdots, b_1 \cdots\}$. We thus naturally turn shape programs into codes with statement types in the first place, following with its parameters, i.e., $[A, a_1, a_2, a_3, B, b_1, b_2, b_3]$. For example, we encode the statement *draw('Top', 'Square', P=(-1,0,0), G=(2,5))* in Figure 1 as $[3, -1, 0, 0, 2, 5, 0, 0]$ where 3 represents the command of drawing square top and the remains are its specific parameters. This way provides a perfect precision to encode and decode shape programs while it cannot handle continuous parameters. Therefore, we proposed an extra *Program*VQVAE in appendix E.3 to handle the case of the continuous parameters. The decoding process is accomplished with the deterministic program parser (Tian et al., 2019).

3.2 SHAPECODE TRANSFORMER

After developing all the encoders and decoders, we can turn Point Clouds, Texts, and Programs into intermediate codes $\mathcal{C}^{point}$, $\mathcal{C}^{text}$, and $\mathcal{C}^{program}$ and reconstruct thereon. ShapeCode Transformer performs over a pair of discrete codes and transforms from one type of code to another type. Similar to (Ramesh et al., 2021), we model the pair of discrete codes as a single data stream. For example, if transforming to point cloud codes from text codes, we model data like $[c^t_1, \cdots, c^t_{N_{text}}, c^p_1, \cdots, c^p_{N_{point}}]$. In training, we pad 0 at the left of data (e.g., $[0, c^t_1, \cdots, c^t_{N_{text}}, \cdots, c^p_{N_{point}}]$) and use full attention masks (Child et al., 2019) to force ShapeCode Transformer autoregressively predicts the next token based on all the seen ones, i.e., predict c_k with seen $[0, \cdots, c_{k-1}]$. Assume $\{c^{gt}_k\}$ is the corresponding ground-truth token, our loss for this transformation would be as the below, where $\mathcal{L}$ is implemented as a cross-entropy loss in our experiments.

$$\sum_k \mathcal{L}(c_k, c^{gt}_k | [0, \cdots, c_{k-1}]) \tag{1}$$

As modeled above, the second type of code gets full attention to the first type of code in predictions. Our modeling can be viewed as maximizing the evidence lower bound on the joint likelihood of the distribution over the input two types of code $p_\theta(\mathcal{C}^{type_i}, \mathcal{C}^{type_j}), type_i \in \{point, text, program\}$, where p_θ is our ShapeCode Transformer implemented by standard Transformer (Vaswani et al., 2017) (Details in Appendix E). Our total training losses are the summation of equation (1) over multiple shape code pairs, where the code pair types include $(\mathcal{C}^{text}, \mathcal{C}^{point})$, $(\mathcal{C}^{point}, \mathcal{C}^{text})$, and $(\mathcal{C}^{point}, \mathcal{C}^{program})$. We introduce Gumble noise in the inference to a set of plausible results for a given input.

One of the challenges in our modeling is that we can generate either texts or programs conditional on the same input point clouds. How do we let ShapeCode Transformer know which type of target code is what we need? Our solution is to encode our pairs of tokens with different positional embedding, i.e., we have two different positional encoding $Pos1([\mathcal{C}^{point}, \mathcal{C}^{text}])$ and $Pos1([\mathcal{C}^{point}, \mathcal{C}^{program}])$. Different positional encodes can guide ShapeCode Transformer to generate desirable target codes.

ShapeCode Transformer communicates shape code with encoders and decoders only. It maintains $[\#tokens, 512]$ embeddings for each type of code $\{\mathcal{C}^{point}, \mathcal{C}^{text}, \mathcal{C}^{program}\}$. For each optimization, once the code is received from the encoder, ShapeCode Transformer indexes the self-maintained embedding through the received code. It transforms shape code based on the indexed embedding and outputs discrete shape code. $\mathcal{C}^{point}$ has $N_{point} = 128$ tokens with a vocabulary size of 512 via argmaxing from the *Point*VQVAE encoder output; $\mathcal{C}^{text}$ has $N_{text} = 256$ tokens of vocabulary size $49,408$, we pad between the last valid text token and the beginning of another token with a value between $[49408, 49408 + 256]$; $\mathcal{C}^{program}$ has $N_{program} = 240$ tokens and a vocabulary size of 78.

4 EXPERIMENTS

We conduct *Text* $\Longrightarrow$ *Point Cloud*, *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* tasks to verify our performance, and *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud* (Shape completion) task to show the extensibility of our framework (Appendix A). For each task, we have two versions of our method: Shape Compiler Limited and Shape Compiler. **Shape Compiler Limited** means that we restrict both *Point*VQVAE and ShapeCode Transformer to train on the same data and task as the baselines for fair comparisons. **Shape Compiler** is our full model trained on all tasks and data to check if joint training on heterogeneous data and tasks leads to improvement. Our used data are briefly described below. More data collection details are listed in Appendix D.

Shape-Text Pairs: To investigate the structural connections between text and shape, we collect a total of 107,371 (*Point Cloud*, *Structure-Related Text*) pairs from a total of 20,355 shapes with 9.47 words per description on average and 26,776 unique words. We use 85% shapes for training and the remaining 15% for testing. Data collection details are listed in Appendix D.

3D Shape Assets: To achieve a general framework, we collect various 3D shapes and obtain a total of 140,419 shapes of 144 categories from ShapeNet (Chang et al., 2015), ABO (Collins et al., 2021), and Program Synthetic (Tian et al., 2019) datasets. For each shape, we sample 10,000 points as inputs and remove all artificial colors and textures to focus on shape geometry and compositionality.

Shape-Program Pairs: We followed (Tian et al., 2019) and synthesized 120,000 (*Point Cloud*, *Program*) pairs. For testing, Tian et al. (2019) sampled shapes from ShapeNet, and we use the same sets to evaluate in our experiments for fair comparisons.

4.1 *Text* $\Longrightarrow$ *Point Cloud*

Neural Shape Compiler can generate point clouds corresponding to text prompt (Figure 3). We compare methods (Jain et al., 2021; Sanghi et al., 2022) that rely primarily on CLIP Radford et al. (2021) to generate 3D shapes and methods that require 3D-text pairs (Chen et al., 2018; Liu et al., 2022b). We use our shape-text training pairs to train CWGAN (Chen et al., 2018) and Shape IMLE diversified model (Liu et al., 2022b), and our Shape Compiler Limited for fair comparisons. Since this paper focuses on geometry, we only compare with (Chen et al., 2018; Liu et al., 2022b) in geometrical aspects, which is the same comparison way used in (Mittal et al., 2022). We compare CLIP-Forge (Sanghi et al., 2022) and DreamField (Jain et al., 2021) to show the gap between exploiting the 2D-text model and training over 3D-text pairs. Since DreamField learns a neural radiance field over a single text with tons of sampling, its training time is too long to test in the scale of our test set (Appendix E.4). We only compare its qualitative results. In order to benchmark *Text* $\Longrightarrow$ *Point Cloud* task, we turn models which output 3D voxels into point clouds by sampling 2,048 points on their output voxels. Shape Compiler is trained with all heterogeneous data and tasks described in Section 4 to show the benefits from jointly training over all tasks.

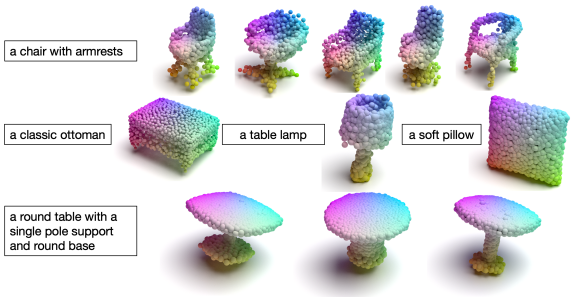


Figure 3: *Text* $\Longrightarrow$ *Point Cloud* results by Neural Shape Compiler. A text prompt may translate to multiple shapes.

Evaluation Protocol: To benchmark *Text* $\Longrightarrow$ *Point Cloud* task, we measure generative performance from multiple angles: quality, fidelity, and diversity. (1) For measuring quality, we use Minimal Matching Distance (MMD) (Achlioptas et al., 2018) to check if the generated shape distributions are close to ground-truth shape distributions by computing the distance between the set of all our generated point clouds and the set of all ground-truth point clouds. We use Chamfer distance for each pair of point clouds in computing MMD. (2) For measuring fidelity, we compute the minimal Chamfer distance between all generated point clouds and the corresponding ground truth shape of the input text. We report the Average Minimal Distance (AMD) over all the test texts. (3) For measuring diversity, we adopt Total Mutual Difference (TMD) (Wu et al., 2020), which computes the difference between all the generated point clouds of the same text inputs. For each generated point cloud P_i, we compute its average Chamfer distance CD_{P_i} to other $k-1$ generated point clouds $P_{j\,j\neq i}$ and compute the average: $TMD = \text{Avg}_{i=1}^{k} CD_{P_i}$. Given an input text, every model finalizes 48 normalized point clouds of 2,048 points.

Results: According to Table 1, Shape Compiler variants generally outperform CLIP-Forge (Sanghi et al., 2022), CWGAN (Chen et al., 2018), and Shape IMLE Diversified (Liu et al., 2022b), especially in AMD (fidelity), which demonstrates our proposed method can generate shapes that more closely match the input texts which contain levels of geometrical details. This is consistent with the qualitative results in Figure 4, where all compared baselines cannot handle the text prompt containing many structural details well. Despite the success of using CLIP in 3D shape generation with text prompt containing simple structure descriptions (Sanghi et al., 2022; Jain et al., 2021), CLIP-Forge shows undesirable AMD and MMD scores in both ShapeGlot and ABO datasets, and DreamField fails to generate structures corresponding to the text prompt, although they are perceptually fine. Also, our experimental results in Appendix B suggest that (1) the text embedding of CLIP models (Radford et al., 2021) may not be able to reflect differences in small changes in the text, which may lead to large changes in structure (e.g., *CosineSimilarity*("*a chair with armrests*", "*a chair with no armrests*") = 0.988); and (2) the cosine similarity between the rendered 3D shape images and the text embedding may not tell its alignment degree if the text prompt and 3D geometry contain details. The above evidence may suggest CLIP is unsuitable for generating shapes with structure details.

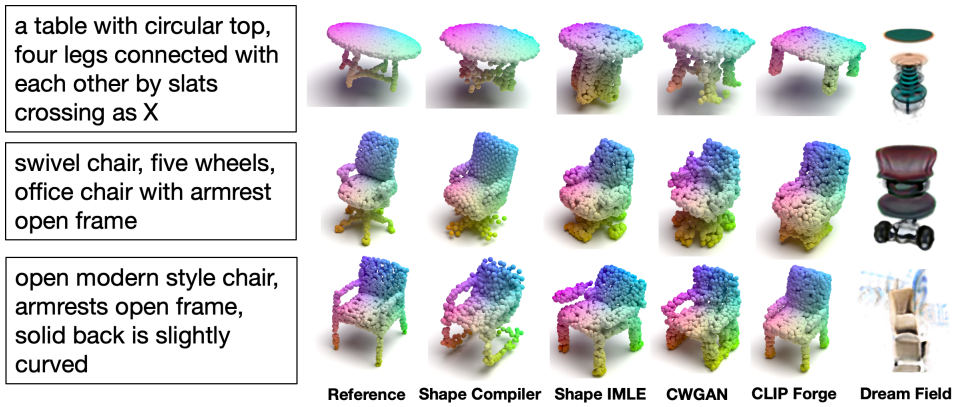


Figure 4: *Text* $\Longrightarrow$ *Point Cloud* results. Neural Shape Compiler can generate corresponding point clouds to the text prompt with structural details, while the baselines generate inaccurate structures or fewer alignments with the text prompt.

One potential drawback in (Chen et al., 2018; Liu et al., 2022b) is the use of 3D convolutional layers, which make their models tend to overfit training data distribution. This is also shown in Figure 4, where their results are less correlated with the input texts. Shape IMLE outperforms CWGAN, consistent with the discovery in (Liu et al., 2022b). However, its inference is much slower than Shape Compiler due to the use of 3D convolutional layers and implementations, whereas our method infers low-dimensional codes and decodes them through a 1D residual decoder. In our same single GPU, Shape IMLE takes 1003.23 seconds to generate 48 shapes, while Shape Compiler takes only 14.69 seconds.

Dataset	Method	MMD ↓	AMD ↓	TMD ↑
ShapeGlot	CWGAN	22.46	27.32	0.67
	Shape IMLE	12.37	14.92	1.83
	CLIP-Forge	36.43	64.5	2.45
	Shape Compiler Limited	6.19	11.27	1.85
	Shape Compiler	5.7	7.31	2.8
Text2shape	CWGAN	10.42	18.21	0.79
	Shape IMLE	6.73	15.34	2.34
	CLIP-Forge	5.16	19.69	1.34
	Shape Compiler Limited	6.19	14.21	1.53
	Shape Compiler	4.53	11.66	3.07
ABO	CWGAN	12.74	22.31	0.52
	Shape IMLE	6.43	13.67	1.42
	CLIP-Forge	7.89	23.35	1.23
	Shape Compiler Limited	4.93	8.33	0.67
	Shape Compiler	4.76	7.77	1.34

Table 1: *Text* $\Longrightarrow$ *Point Cloud* performance. MMD (quality), AMD (fidelity), and TMD (diversity) are multiplied by 10^3, 10^3 and 10^2, respectively.

Shape Compiler consistently outperforms Shape Compiler Limited indicating the proposed framework benefits from the jointly training over all heterogeneous data and tasks. Despite higher scores shown in Table 1, due to the use of point cloud representation, the inherent difficulty of 3D shape generation with structure details, and the introduction of Gumbel noise, our framework produces rough boundaries, uneven surfaces, and irrelevant shapes. There is a lot of room for research in the proposed framework and *Text* $\Longrightarrow$ *Point Cloud* task.

4.2 *Point Cloud* $\Longrightarrow$ *Text*

Neural Shape Compiler performs *Point Cloud* $\Longrightarrow$ *Text* tasks that describe given shape point clouds as shown in Figure 5. To compare, we followed (Chen et al., 2021) and implemented a 3DEncoder-LSTM model as our baseline by replacing the image encoder of Show-Attend-Tell(Vinyals et al., 2015) with a 3D encoder. We adopt a similar encoder structure as *Point*VQVAE and set the output feature to be 512 dimension, the same value as our intermediate representations. An LSTM model will then be trained over the 512 output features to predict words gradually, and an end word will be the last token. For each shape, all methods will generate 48 descriptions to compare.

Evaluation Protocol: We measure the quality of the generated shape descriptions with our annotations in validation sets. Here we adopt the common metrics used in text generations tasks (Vinyals et al., 2015; Koncel-Kedziorski et al., 2019) including BLEU (4-gram) (Papineni et al., 2002), CIDER (Vedantam et al., 2015), and ROUGE (Lin, 2004). We also adopt Dist-1 and Dist-2 in (Li et al., 2015) to measure the diversity of the generated results: Dist-1 and Dist-2 are the results of distinct unigrams and bigrams divided by the total number of tokens (maximum 100 in our table).

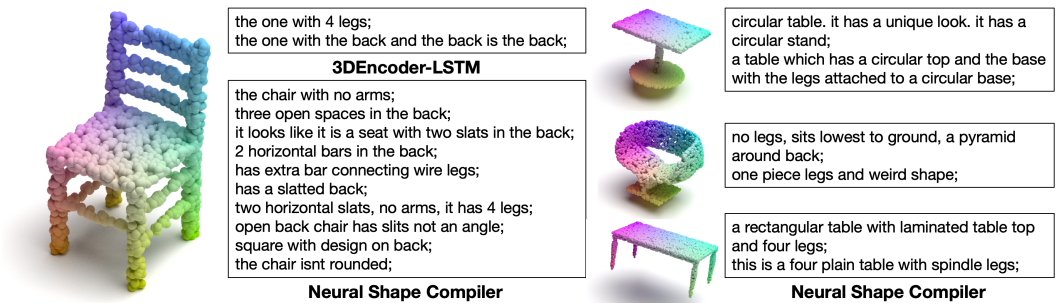


Figure 5: *Point Cloud* $\Longrightarrow$ *Text* results. One description per sentence. The shown shapes are from test sets, and Neural Shape Compiler tell their structures well.

Results: According to Dist-1 and Dist-2 in Table 2, Shape Compiler generates much more diverse captions than 3DEncoder-LSTM (Vinyals et al., 2015; Chen et al., 2021) due to the help of involving Gumbel noise over the learned priors in the inference. This is also shown in Figure 5 where Shape Compiler can describe given shapes in multi-angles.

Also, our framework achieves generally higher BLEU, CIDER, and ROUGE scores than 3DEncoder-LSTM. By inspecting the predictions, we found that 3DEncoder-LSTM tends to predict the frequent words in datasets (e.g., back and legs), and the output descriptions are less meaningful in semantics. However, those repetitive words can help them achieve higher BLEU, CIDER and ROUGE scores in Text2Shape and ShapeGlot. Therefore, evaluating *Point Cloud* $\Longrightarrow$ *Text* performance with both quality and diversity metrics is essential. Furthermore, the ground-truth descriptions differ significantly from our predictions on diversity metrics, suggesting that the proposed method still has huge gaps in achieving human-level shape captioning.

Dataset	Method	BLEU-4 ↑	CIDER ↑	ROUGE ↑	Dist-1 ↑	Dist-2 ↑
ShapeGlot	3DEncoder-LSTM	2.78	2.93	18.61	0.016	0.017
	Shape Compiler Limited	2.96	2.15	22.26	1.174	14.146
	Shape Compiler	3.89	3.35	22.78	1.698	17.764
	Ground-Truth	-	-	-	6.908	37.92
Text2Shape	3DEncoder-LSTM	1.01	0.84	21.18	0.027	0.042
	Shape Compiler Limited	2.96	1.5	25.97	0.792	9.984
	Shape Compiler	3.02	1.52	25.52	0.828	10.109
	Ground-Truth	-	-	-	5.906	34.05
ABO	3DEncoder-LSTM	0.01	1.39	2.98	0.031	0.047
	Shape Compiler Limited	8.95	111.28	26.89	3.605	5.082
	Shape Compiler	10.03	127.82	28.98	3.968	24.176
	Ground-Truth	-	-	-	18.294	49.097

Table 2: *Point Cloud* $\Longrightarrow$ *Text* performance. BLEU, CIDER, and ROUGE measure quality. Dist-1 and Dist-2 measure diversity.

4.3 *Point Cloud* $\Longrightarrow$ *Program*

Shape Compiler performs transformation: *Point Clouds* $\Rightarrow$ *Program*, which can help us understand how the point clouds are assembled by parts and their regularities. For example, in Figure 6, we can tell how to decompose the point clouds into primitives via the symbolic words in the programs and also find the mapping between points and the corresponding primitive via executing the program. Furthermore, the symmetry relationship between those racks of the cabinet is implicitly encoded in the *FOR* statements of the program. Neural Program Generator (Tian et al., 2019) is our most direct baseline. They used a two-layered LSTM to gradually generate program blocks and programs inside each block to form the final shape of programs. We will not adopt the guided adaptation used in (Tian et al., 2019) in our experiments because it needs extra training loops and prevents the practical use of program generation (further discussion is included in Appendix G). We also compare with CSGNet (Sharma et al., 2018), which applies boolean operations on shape primitives and assemble shape recursively.

Evaluation Protocol: Shape Program (Tian et al., 2019), CSGNet (Sharma et al., 2018), and our models will predict programs under specific domain languages. By executing the output programs, we can obtain volumetric representation shapes; then, we can compute IoU based on the output voxels. Also, we sample points over the surface of output voxels for computing Chamfer distance. Due to the benefits of being a probabilistic method, our framework can predict many programs given

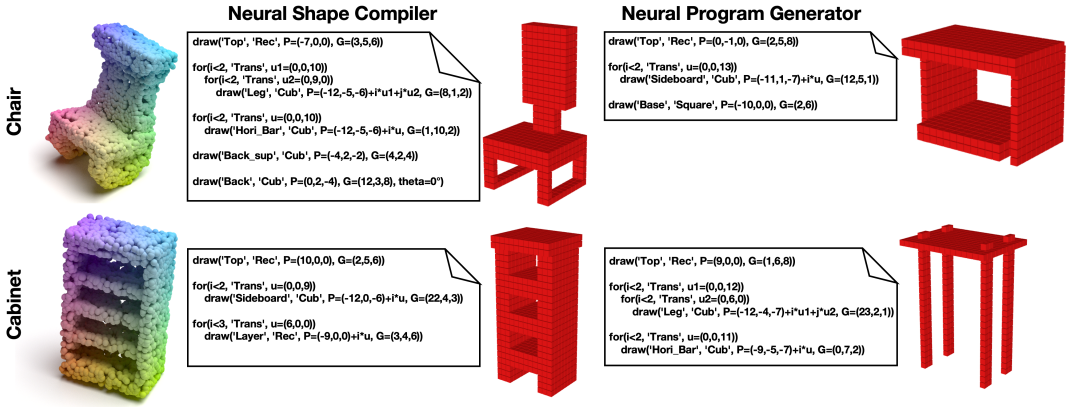


Figure 6: *Point Cloud* $\Longrightarrow$ *Program* results. Neural Shape Compiler can well infer programs for the shown shapes of unseen categories and reconstruct shapes in a creative manner (e.g., represent the semicircle leg with bars).

a single point cloud. We here generate 48 programs for comparisons, the same number we used in our text tasks. Ablation studies of the generated program numbers are included in Appendix G.

Result: According to Table 3, Shape Compiler outperforms all the methods due to the benefits of multimodal learning on all tasks and data. It also suggests our framework can effectively take advantage of task-unrelated heterogeneous data. Figure 6 shows two positive examples of our framework that better handles novel data than the baseline. Both cases show that the compared baseline mainly relies on memorizing the training table data, while Shape Compiler successfully predicts the rough structure and the regularities. Regarding the scores, Shape Compiler Limited achieved comparable performance with Shape Generator, while the significant performance gap in cabinet and bench categories shows our framework has stronger generalizability. Due to the limitation of the current program grammar, our method cannot handle complex structures for now. We raise attention to the next-level shape program, and some further discussion is included in Appendix G.

Metric	Method	Chair	Table	Bed	Sofa	Cabinet	Bench	Avg
IoU ↑	CSGNet (Sharma et al., 2018)	0.365	0.406	-	-	-	-	-
	Program Generator (Tian et al., 2019)	0.438	0.517	0.254	0.324	0.304	0.216	0.342
	Shape Compiler Limited	0.429	0.539	0.27	0.31	0.504	0.314	0.394
	Shape Compiler	0.47	0.631	0.264	0.419	0.524	0.389	0.45
CD ↓	CSGNet (Sharma et al., 2018)	7.73	7.24	-	-	-	-	-
	Program Generator (Tian et al., 2019)	1.64	1.97	4.78	3.14	2.95	2.71	2.87
	Shape Compiler Limited	1.53	1.21	4.51	2.84	1.58	1.57	2.21
	Shape Compiler	1.27	0.92	4.39	2.52	1.49	1.62	2.04

Table 3: *Point Cloud* $\Longrightarrow$ *Program* performance. CD is multiplied by 10^2. Avg denotes average number among categories.

5 Conclusion

We proposed Neural Shape Compiler to translate between three shape abstractions: *Text*, *Point Cloud*, and *Program*. With the help of *Point*VQVAE, it achieved great performance in *Text* $\Longrightarrow$ *Point Cloud*, *Point Cloud* $\Longrightarrow$ *Text*, *Point Cloud* $\Longrightarrow$ *Program*, and PointCloud Completion tasks via a unified and extendable framework. Also, our experiments show that Neural Shape Compiler benefits from joint training over all heterogeneous data and tasks. Despite showing promises, Shape Compiler has limitations and draws attention to several related directions for future research (Appendix C). We hope that Neural Shape Compiler can provide an effective framework for connecting different shape abstractions and facilitate the progress in 3D multimodal learning research.

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pp. 40–49. PMLR, 2018.

Panos Achlioptas, Judy Fan, Robert XD Hawkins, Noah D Goodman, and Leonidas J Guibas. Shapeglot: Learning language for shape differentiation. *arXiv preprint arXiv:1905.02925*, 2019.

Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc Van Gool. Soft-to-hard vector quantization for end-to-end learned compression of images and neural networks. *arXiv preprint arXiv:1704.00648*, 3, 2017.

Alfred Aho and Jeffrey Ullman. Abstractions, their algorithms, and their compilers. *Communications of the ACM*, 65(2):76–91, 2022.

Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. *Compilers: principles, techniques, & tools*. Pearson Education India, 2007.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.

Andrew W Appel. *Modern compiler implementation in C*. Cambridge university press, 2004.

Song Bai, Xiang Bai, Zhichao Zhou, Zhaoxiang Zhang, and Longin Jan Latecki. Gift: A real-time and scalable 3d shape search engine. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5023–5032, 2016.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Dimitri P Bertsekas and David A Castanon. The auction algorithm for the transportation problem. *Annals of Operations Research*, 20(1):67–96, 1989.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Michael M Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pp. 1704–1711. IEEE, 2010.

Gülen Çağdaş. A shape grammar: the language of traditional turkish houses. *Environment and Planning B: Planning and Design*, 23(4):443–464, 1996.

Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *European Conference on Computer Vision*, pp. 364–381. Springer, 2020.

Roberto Calandra, Andrew Owens, Dinesh Jayaraman, Justin Lin, Wenzhen Yuan, Jitendra Malik, Edward H Adelson, and Sergey Levine. More than a feeling: Learning to grasp and regrasp using vision and touch. *IEEE Robotics and Automation Letters*, 3(4):3300–3307, 2018.

Peter Calingaert. *Assemblers, compilers, and program translation*. WH Freeman & Co., 1979.

Dan Cascaval, Mira Shalah, Phillip Quinn, Rastislav Bodik, Maneesh Agrawala, and Adriana Schulz. Differentiable 3d cad programs for bidirectional editing. In *Computer Graphics Forum*, volume 41, pp. 309–323. Wiley Online Library, 2022.

Angel Chang, Manolis Savva, and Christopher D Manning. Learning spatial knowledge for text to 3d scene generation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2028–2038, 2014.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Siddhartha Chaudhuri, Daniel Ritchie, Jiajun Wu, Kai Xu, and Hao Zhang. Learning generative models of 3d structures. In *Computer Graphics Forum*, volume 39, pp. 643–666. Wiley Online Library, 2020.

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*, pp. 100–116. Springer, 2018.

Zhenyu Chen, Ali Gholami, Matthias Nießner, and Angel X Chang. Scan2cap: Context-aware dense captioning in rgb-d scans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3193–3203, 2021.

An-Chieh Cheng, Xueting Li, Sifei Liu, Min Sun, and Ming-Hsuan Yang. Autoregressive 3d shape generation via canonical mapping. *arXiv preprint arXiv:2204.01955*, 2022.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Adam Coates and Andrew Ng. Selecting receptive fields in deep networks. *Advances in neural information processing systems*, 24, 2011.

Jasmine Collins, Shubham Goel, Achleshwar Luthra, Leon Xu, Kenan Deng, Xi Zhang, Tomas F Yago Vicente, Himanshu Arora, Thomas Dideriksen, Matthieu Guillaumin, et al. Abo: Dataset and benchmarks for real-world 3d object understanding. *arXiv preprint arXiv:2110.06199*, 2021.

Theo Deprelle, Thibault Groueix, Matthew Fisher, Vladimir Kim, Bryan Russell, and Mathieu Aubry. Learning elementary structures for 3d shape generation and matching. *Advances in Neural Information Processing Systems*, 32, 2019.

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.

Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 605–613, 2017.

Yifan Feng, Zizhao Zhang, Xibin Zhao, Rongrong Ji, and Yue Gao. Gvcnn: Group-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 264–272, 2018.

Rao Fu, Xiao Zhan, Yiwen Chen, Daniel Ritchie, and Srinath Sridhar. Shapecrafter: A recursive text-conditioned 3d shape generation model. *arXiv preprint arXiv:2207.09446*, 2022.

Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. In *Advances In Neural Information Processing Systems*, 2022a.

Ruohan Gao, Zilin Si, Yen-Yu Chang, Samuel Clarke, Jeannette Bohg, Li Fei-Fei, Wenzhen Yuan, and Jiajun Wu. Objectfolder 2.0: A multisensory object dataset for sim2real transfer. *arXiv preprint arXiv:2204.02389*, 2022b.

Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4857–4866, 2020.

T Groueix, M Fisher, VG Kim, BC Russell, and M Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. arxiv 2018. *arXiv preprint arXiv:1802.05384*, 1802.

Armin Gruen and Devrim Akca. Least squares 3d surface and curve matching. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59(3):151–174, 2005.

Zhizhong Han, Honglei Lu, Zhenbao Liu, Chi-Man Vong, Yu-Shen Liu, Matthias Zwicker, Junwei Han, and CL Philip Chen. 3d2seqviews: Aggregating sequential views for 3d global feature learning by cnn with hierarchical attention aggregation. *IEEE Transactions on Image Processing*, 28(8):3986–3999, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spacy: Industrial-strength natural language processing in python. 2020.

Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Jun-Xiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R Martin. Subdivision-based mesh convolution networks. *ACM Transactions on Graphics (TOG)*, 41(3):1–16, 2022.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

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.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Mengqi Ji, Juergen Gall, Haitian Zheng, Yebin Liu, and Lu Fang. Surfacenet: An end-to-end 3d neural network for multiview stereopsis. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2307–2315, 2017.

R Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J Mitra, and Daniel Ritchie. Shapeassembly: Learning to generate programs for 3d shape structure synthesis. *ACM Transactions on Graphics (TOG)*, 39(6):1–20, 2020.

R Kenny Jones, David Charatan, Paul Guerrero, Niloy J Mitra, and Daniel Ritchie. Shapemod: macro operation discovery for 3d shape programs. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. Text generation from knowledge graphs with graph transformers. *arXiv preprint arXiv:1904.02342*, 2019.

Alon Lahav and Ayellet Tal. Meshwalker: Deep mesh understanding by random walks. *ACM Transactions on Graphics (TOG)*, 39(6):1–13, 2020.

Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pp. 75–86. IEEE, 2004.

Aoxue Li, Tiange Luo, Zhiwu Lu, Tao Xiang, and Liwei Wang. Large-scale few-shot learning: Knowledge transfer with class hierarchy. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7212–7220, 2019.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*, 2015.

Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pp. 74–81, 2004.

Hsueh-Ti Derek Liu, Vladimir G Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. Neural subdivision. *arXiv preprint arXiv:2005.01819*, 2020a.

Minghua Liu, Lu Sheng, Sheng Yang, Jing Shao, and Shi-Min Hu. Morphing and sampling network for dense point cloud completion. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 11596–11603, 2020b.

Zhengzhe Liu, Peng Dai, Ruihui Li, Xiaojuan Qi, and Chi-Wing Fu. Iss: Image as stetting stone for text-guided 3d shape generation. *arXiv preprint arXiv:2209.04145*, 2022a.

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

Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Shitong Luo and Wei Hu. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2837–2845, 2021.

Tiange Luo, Tianle Cai, Mengxiao Zhang, Siyu Chen, Di He, and Liwei Wang. Defective convolutional networks. *arXiv preprint arXiv:1911.08432*, 2019.

Tiange Luo, Kaichun Mo, Zhiao Huang, Jiarui Xu, Siyu Hu, Liwei Wang, and Hao Su. Learning to group: A bottom-up framework for 3d part discovery in unseen categories. *arXiv preprint arXiv:2002.06478*, 2020.

Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928. IEEE, 2015.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

Oscar Michel, Roi Bar-On, Richard Liu, Sagie Benaim, and Rana Hanocka. Text2mesh: Text-driven neural stylization for meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13492–13502, 2022.

Niloy J Mitra and An Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pp. 322–328, 2003.

Niloy J Mitra, Leonidas J Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM Transactions on Graphics (TOG)*, 25(3):560–568, 2006.

Niloy J Mitra, Leonidas J Guibas, and Mark Pauly. Symmetrization. *ACM Transactions on Graphics (TOG)*, 26 (3):63–es, 2007.

Paritosh Mittal, Yen-Chi Cheng, Maneesh Singh, and Shubham Tulsiani. Autosdf: Shape priors for 3d completion, reconstruction and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 306–315, 2022.

Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *arXiv preprint arXiv:1908.00575*, 2019.

Luca Morreale, Noam Aigerman, Paul Guerrero, Vladimir G Kim, and Niloy J Mitra. Neural convolutional surfaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19333–19342, 2022.

Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7588–7597, 2019.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 165–174, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Or Patashnik, Zongze Wu, Eli Shechtman, Daniel Cohen-Or, and Dani Lischinski. Styleclip: Text-driven manipulation of stylegan imagery. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2085–2094, 2021.

Mark Pauly, Niloy J Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J Guibas. Discovering structural regularity in 3d geometry. In *ACM SIGGRAPH 2008 papers*, pp. 1–11. 2008.

Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

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*, pp. 652–660, 2017a.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pp. 5099–5108, 2017b.

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*, pp. 8748–8763. PMLR, 2021.

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*, pp. 8821–8831. PMLR, 2021.

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.

Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 693–700. JMLR Workshop and Conference Proceedings, 2010.

Aditya Sanghi, Hang Chu, Joseph G Lambourne, Ye Wang, Chin-Yi Cheng, Marco Fumero, and Kamal Rahimi Malekshan. Clip-forge: Towards zero-shot text-to-shape generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18603–18613, 2022.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5515–5523, 2018.

Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. *arXiv preprint arXiv:1901.02875*, 2019.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, 33:19667–19679, 2020.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

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

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.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4566–4575, 2015.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, 2015.

Cheng Wang, Ming Cheng, Ferdous Sohel, Mohammed Bennamoun, and Jonathan Li. Normalnet: A voxel-based cnn for 3d object classification and retrieval. *Neurocomputing*, 323:139–147, 2019.

Yanzhen Wang, Kai Xu, Jun Li, Hao Zhang, Ariel Shamir, Ligang Liu, Zhiquan Cheng, and Yueshan Xiong. Symmetry hierarchy of man-made objects. In *Computer graphics forum*, volume 30, pp. 287–296. Wiley Online Library, 2011.

Niklaus Wirth, Niklaus Wirth, Niklaus Wirth, Suisse Informaticien, and Niklaus Wirth. *Compiler construction*, volume 1. Citeseer, 1996.

Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems*, 29, 2016.

Rundi Wu, Xuelin Chen, Yixin Zhuang, and Baoquan Chen. Multimodal shape completion via conditional generative adversarial networks. In *European Conference on Computer Vision*, pp. 281–296. Springer, 2020.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

Kai Xu, Hao Zhang, Wei Jiang, Ramsay Dyer, Zhiquan Cheng, Ligang Liu, and Baoquan Chen. Multi-scale partial intrinsic symmetry detection. *ACM Transactions On Graphics (TOG)*, 31(6):1–11, 2012.

Xingguang Yan, Liqiang Lin, Niloy J Mitra, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Shapeformer: Transformer-based shape completion via sparse representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6239–6249, 2022.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4541–4550, 2019.

Alon Zakai. Emscripten: an llvm-to-javascript compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 301–312, 2011.

Renrui Zhang, Ziyu Guo, Wei Zhang, Kunchang Li, Xupeng Miao, Bin Cui, Yu Qiao, Peng Gao, and Hongsheng Li. Pointclip: Point cloud understanding by clip. *arXiv preprint arXiv:2112.02413*, 2021.

Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Josh Tenenbaum, Bill Freeman, and Jiajun Wu. Learning to reconstruct shapes from unseen classes. *Advances in neural information processing systems*, 31, 2018.

Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5826–5835, 2021.

Appendix

Table of Contents

A Empower 3D Methods

With the help of Neural Shape Compiler, we may better share the progress across different tasks and shape abstractions. In this section, we show the two examples mentioned in the introduction: (1) turn the success of shape completion into more accurate shape structural understanding; (2) obtain hierarchical information for the single image reconstruction result.

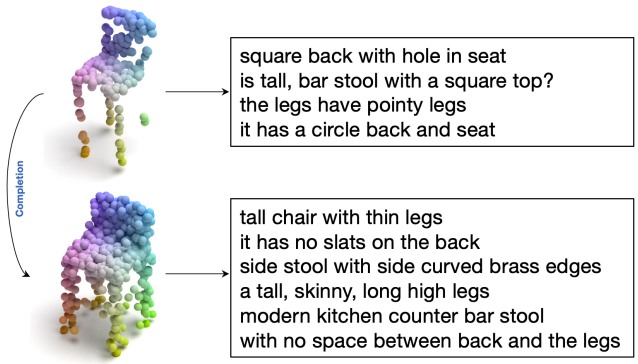
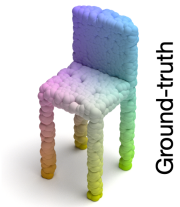


Figure 7: The success of shape completion boosts shape structural understandings. We use Neural Shape Compiler to do *Point Cloud* $\Longrightarrow$ *Text* for both the partial chair and the complete chair. The ground-truth shape is on the top right.



Method	Chair	Airplane	Car
PVD (Zhou et al., 2021)	7.34	1.19	3.83
Shape Compiler Limited	7.57	1.62	4.82
Shape Compiler	6.9	1.56	3.81

Table 4: *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud*. Numbers are Chamfer distance $\downarrow$ with multiplying 10^2.

In Figure 7, we individually perform *Point Cloud* $\Longrightarrow$ *Text* over the partial chair and the complete chair. The ground-truth chair is on the right. For the partial chair, the captioning results include hallucinated descriptions, like *hole in the back* and *circle back*, and also accurate descriptions of the partial chair but not precisely related to the ground-truth one, like *pointy legs*. Compared to it, the descriptions for the completion result are more related to our ground-truth point clouds.

Beyond the qualitative result, we also test our Neural Shape Compiler in *Partial Point Cloud* $\Longrightarrow$ *Complete Point Cloud* (shape completion) task by projecting both partial and full point clouds into discrete codes and performing conditional generation thereon, i.e. (*Partial PointCloud*, *Complete PointCloud*). We followed the benchmark provided in (Zhang et al., 2018; Zhou et al., 2021) and the detailed data preparation is listed in Appendix D.3. We train models on all training shapes in a class-agnostic way. Results in Table 4 show Shape Compiler achieved great performance compared with the previous method. Also, PVD (Zhou et al., 2021) is much slower than Shape Compiler because they involve very long steps (1,000) in the diffusion process, while our code prediction process is much more effective.

Also, we show Neural Shape Compiler can help obtain hierarchical information for the reconstruction results from a single image. As shown in Figure 8, we first use GenRe (Zhang et al., 2018) to reconstruct the 3D shape from the provided single-view image and then apply Neural Shape Compiler thereon to generate its corresponding shape programs and texts for obtaining assemblies, regularities, and semantic meanings.

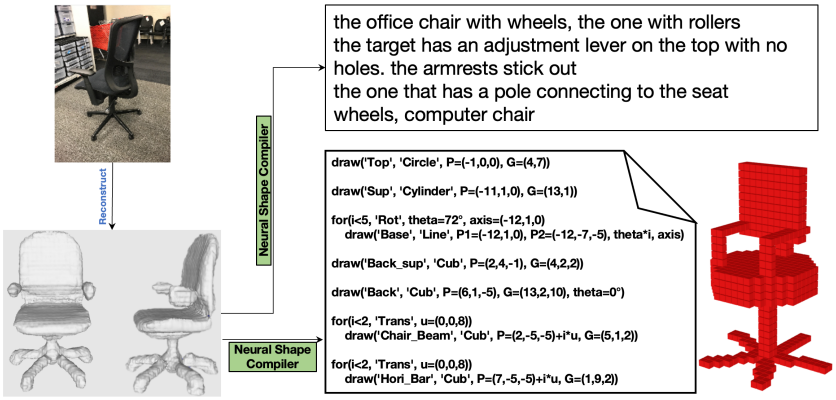


Figure 8: Example: Neural Shape Compiler helps obtain hierarchical information for the reconstruction result of a single image, including its structural descriptions, regularities, and how to assemble it.

B WHY CLIP MAY NOT BE SUITABLE FOR GENERATING 3D SHAPES WITH STRUCTURAL DETAILS?

In this section, we empirically study the connections between the learning representation of CLIP (Radford et al., 2021) and 3D shape generation, especially in geometrical aspects. Our results suggest that the text embedding of CLIP models (Radford et al., 2021) may not be able to reflect differences in small changes in the text, which may lead to large changes in the structure; and (2) the inner product between the rendered 3D shape image embedding and the text embedding may not tell its alignment degree if the text prompt and 3D geometry contain details.

We first study the similarity of CLIP text embedding with structural descriptions as shown in Table 5. We computed the cosine similarity between two texts with minor differences that led to huge changes in target 3D shapes. The cosine similarity is defined as equation (1), where f_1 and f_2 are two embedding with the same feature dimension, and ϵ is a very small value to avoid division by zero (we used 1e-6).

$$\text{Similarity} = \frac{f_1 \cdot f_2}{\max\left(\|f_1\|_2 \cdot \|f_2\|_2, \epsilon\right)} \tag{2}$$

The CLIP model we used to compute scores is *ViT-B/32*, while the phenomena are similar using other CLIP models with different backbones. The texts in the second column are similar to those in the

first but will lead to significant structural changes. However, they have very high similarity scores in CLIP text embeddings (almost 1, which is the upper bound for cosine similarity). This may illustrate why CLIP-Forge (Sanghi et al., 2022) achieved pretty low AMD scores (Table 1) in our datasets, where our 3D-text datasets contain many complex structural descriptions as shown in Appendix D. The CLIP text embedding cannot well tell the difference between texts with structural details, which is critical for 3D shape generation.

Text1	Text2	Similarity
a chair with armrests	a chair with no armrests	0.983
a chair with armrests	a chair without armrests	0.988
a swivel chair with armrests	a swivel chair with no armrests	0.979
a swivel chair with armrests	a swivel chair without armrests	0.986
a chair with armrests	a chair with armrests and curved back	0.962
a chair with armrests	a chair with no armrests and curved back	0.965
a swivel chair with armrests	a swivel chair with armrests and curved back	0.981
a swivel chair with armrests	a swivel chair with no armrests and curved back	0.982
a table with circular top	a table with square top	0.932
a table with circular top	a table with rectangular top	0.938
a stool chair with two legs	a stool chair with three legs	0.992
a couch with two seats	a couch with three seats	0.986

Table 5: The rightmost column is the cosine similarity between the CLIP text embeddings (Radford et al., 2021) of the left two texts. The red texts are the highlighted difference for each line.

Also, we study the association between the embedding of the 3D shape rendered images and the text embedding. We use PyTorch3D (Ravi et al., 2020) render 3D shapes into images as shown in the left of Figure 9. We will render each shape with six camera positions and two different point lights. The rendered images are resize to $(224, 224)$ and pass through normalization layer with $mean = (0.48145466, 0.4578275, 0.40821073)$ and $std = (0.26862954, 0.26130258, 0.27577711)$. Then, we obtain each image embedding by using CLIP image branch encoder ($clip.encode\_image$). Each shape has several text descriptions. We tokenize those texts with the tokenizer provided by CLIP and obtain our text embedding by feeding them into CLIP text encoder ($clip.encode\_text$). In Figure 9, the score above each shape is the average cosine similarity (equation 1) between 1 image and each text shown at the top. The clip model we used here is still *ViT-B/32*, and other CLIP models show similar trends.

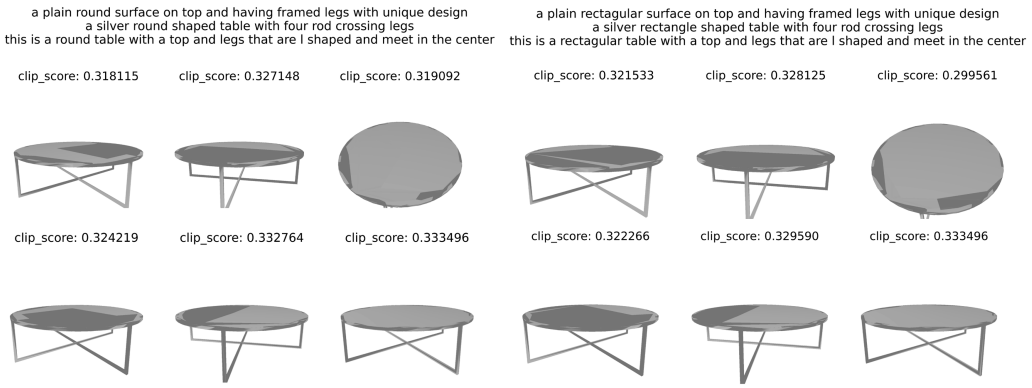


Figure 9: Render the shape with six different camera positions and two different point lights. The texts shown at the top left are the shape-associated texts provided in our datasets, which only contain Round or Circular. We modified them into Rectangle or Rectangular as shown in the top right. The number above each image is the average cosine similarity between CLIP image embedding and each text embedding, where six left numbers are computed with the top left text and six right numbers are computed with the top right text.

We select shapes from modified Text2Shape (Chen et al., 2018) with texts contain Round or Circular and not include Rectangle and Rectangular. Then, we compute the cosine similarity between the rendered image embedding and two different sets of text embedding: (1) the original ground-truth texts which contain Round or Circular; (2) we replace Round or Circular into Rectangle and Rectangular, and obtain texts which not contain any Round or Circular, but have Rectangle and Rectangular. An example of replacement is shown in Figure 9 where the right figure top texts are the replaced ones. We compute the scores over 1,358 shapes with a total of 6,697 texts. As described above, we have six images for each shape, resulting in 40,182 scores. We compute the average and standard deviation over the scores and obtain the score of the first set which only contain Round or Circular 0.2925 ± 0.02173 and the score of the second set which only contain Rectangle or Rectangular 0.2917 ± 0.02211. Although there are huge differences between the two sets of texts, their average cosine similarity is very close. Based on the number, we can tell the similarity between the rendered images CLIP embedding and texts CLIP embedding cannot well demonstrate their alignment degree. The reason may be that the rendered images are outside the training distribution of CLIP, and our texts are not included in CLIP training data. But, both how to render 3D shapes into the training distribution of CLIP and the range of texts included in CLIP datasets are unknown.

The above two experiments may suggest the current CLIP cannot tell structural text description differences and the alignments between 3D rendered images and texts. CLIP may not be suitable for generating 3D shapes with structural details, despite the success of CLIP in image-text tasks Ramesh et al. (2022).

C Limitations and Future Works

Our current framework presents some positive signals, while there are still a lot of things to be considered:

1. Our framework is well extensible, where we showed a case in shape completion task (Table 4). Similarly, we can project images (Van Den Oord et al., 2017), tactile information (Gao et al., 2022b), and other modalities into codes and integrate them into our framework to develop related 3D applications thereon, such as multiview reconstruction (Ji et al., 2017) and multi-modal grasping (Calandra et al., 2018).
2. Neural Shape Compiler currently compiles different shape abstractions. Compared to this, an orthogonal direction is to compile operations on one shape abstraction into practical steps in other abstractions and change it accordingly. For example, we add "*armless*" into a text description of a chair with two arms. This change can be effectively compiled into operations "*remove two arm parts and their symmetry relationship*" in the shape hierarchy tree and "*delete all points belonging to the two arms*" in point clouds.
3. Our approach currently adopts the domain-specific language (DSL) proposed in (Tian et al., 2019) as one of the input and target shape abstractions. However, the DSL's grammar and syntax are not powerful enough to handle various complex shapes. For example, it fails to perform accurate reconstruction or completion for 3D shapes with geometrical details (Chaudhuri et al., 2020) and detect self-symmetries possessed by individual parts (Mitra et al., 2006). Those limitations also limit the capabilities of the proposed framework. Since the progress of shape programs is orthogonal to the proposed framework, the development of shape programs can further benefit our framework. Therefore, this paper also raises the attention to developing the next level of shape programs.
4. Unlike the 3D-text works that leverage the 2D-text progress (e.g., CLIP (Jain et al., 2021; Sanghi et al., 2022)), our paper tried to understand and model the connections between 3D shapes and texts directly. However, this requires large-scale annotated paired data, which is hard to obtain in 3D shapes and a significant limitation for the kinds of methods (Chen et al., 2018; Liu et al., 2022b).
5. Due to the limitation of data, the proposed framework currently cannot well handle out-of-distribution geometry, such as the dragon in (Morreale et al., 2022) and the Hilbert cube in (Liu et al., 2020a). Figure 10 shows the results of *Point Cloud* $\Longrightarrow$ *Text* and shape-to-program over the Hilbert cube by Shape Compiler. The current programs can only tell the coarse geometry, and captions are about similar objects in our training data, such as cubes and cabinets.

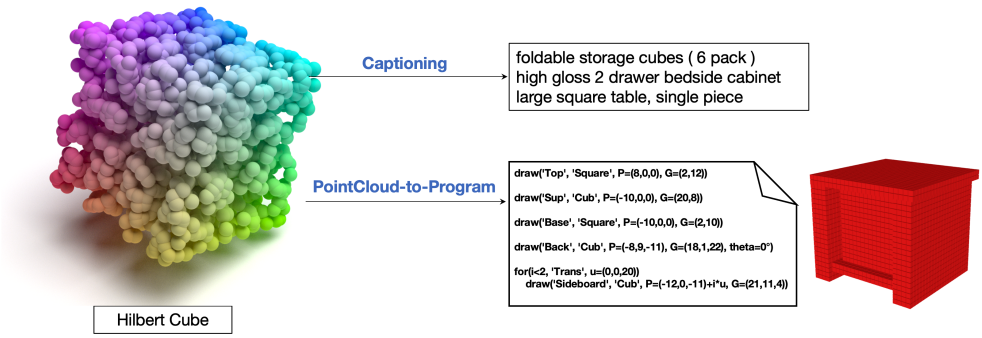


Figure 10: *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* results over the input Hilbert cube point clouds by Shape Compiler. One description per sentence.

6. Our current framework and data focused on correctly modeling the connections between 3D geometry and texts. The experimental results show positive signals that our model can generate geometry corresponding to input texts, captions corresponding to input point clouds, and programs corresponding to input geometry. But, our approach currently does not consider colors, textures, and materials, which will be studied in the future with the support of the proper data.
7. DALL-E (Ramesh et al., 2021) will rank their outputs of (*Text*, *Image*) based on the CLIP scores (Radford et al., 2021). For our tasks, we can rank (*Point Cloud*, *Shape Program*) and (*Partial Point Cloud*, *Complete Point Cloud*) based on auto-encoding performance. However, we cannot rank (*Text*, *Point Cloud*) and (*Point Cloud*, *Text*) due to we lack 3D CLIP models. A straightforward way is rendering Point Clouds into images and leveraging the 2D CLIP model(Radford et al., 2021). However, as shown in Appendix B, the 2D CLIP model failed to capture the connections between texts and 3D structures to some levels. There may be other fundamental ways in 3D to solve the 3D-text ranking problem, and we leave them for future studying.
8. We currently adopt MMD, AMD, and TMD to measure the quality, fidelity, and diversity of the *Text* $\Longrightarrow$ *Point Cloud* results, respectively. However, we do not have proper metrics to measure how realistic the generated shapes are, such as FID used in image generation (Brock et al., 2018). Also, our current metrics can not guarantee the local geometry properties, such as the normal and principal curvature (assuming we re-meshed the point cloud). We probably need efforts to develop the mentioned metrics in the future.
9. Currently, the order of the code sequence is the concatenation order of the part codes obtained by feeding part embedding into the Codebook. The part embedding order is determined by Farthest Point Sampling (FPS) in the centroid selection phase. Therefore, there is no consistent order for the point cloud codes. The inconsistent order will not affect the decoder's performance since its outputs are irrelevant to the concatenation order. For ShapeCode Transformer, it is only required to output the correct tokens in the final vector regardless of the position due to our order-invariant decoder. Therefore, the inconsistent code order might not harm the transformation process much. But still, we should study here more in future works.

D Data Collection

D.1 3D Shape Assets

As mentioned in Section 4, we collect a total of 140,419 shapes from ShapeNet (Chang et al., 2015), ABO dataset (Collins et al., 2021), and Shape Program (Tian et al., 2019). For ShapeNet, we use the most recent release version (ShapeNetCore.v2), which contains 52,472 shapes from 55 categories. ABO dataset contains 7,947 shapes from 98 categories and has nine overlapped shape categories with ShapeNet. We adopt the 4 Chair and 10 Table templates in Shape Program (Tian et al., 2019) to synthesize 40,000 shapes in each category. We do not distinguish shape categories during training for

both *Point*VQVAE and ShapeCode Transformer. We will normalize the input point clouds into a unit ball for addressing dataset gaps.

D.2 SHAPE-TEXT PAIRS

We collect 107,371 (*Point Cloud*, *Structure-Related Text*) pairs by adjusting and annotate the current datasets, including Text2Shape (Chen et al., 2018), ShapeGlot (Achlioptas et al., 2019), and ABO (Collins et al., 2021) datasets. We take tables from Text2Shape, chairs from ShapeGlot, and all shapes from the ABO dataset. Therefore, our dataset consists of furniture, most of which are tables and chairs. We list our considerations and data collection details below.

Text2Shape: (Chen et al., 2018) provides descriptions for both chairs and tables. Through careful inspection, we found there are a large number of descriptions regarding artificial colors (e.g., red, blue), textures (e.g., green grids on the top), and materials (e.g., wooden, steel). Since this work aims to focus on the 3D geometry and will not predict any colors for output point clouds, we delete those artificial texture contents but remain the geometrical ones. Also, compared to Text2Shape, ShapeGlot (Achlioptas et al., 2019) provides much more geometrical descriptions for chairs, where the chairs have a very high overlap with chairs in Text2Shape. We thus discard the chair descriptions and only adopt the table descriptions of Text2Shape in our data. For a few data which lack structural descriptions, we will annotate them manually. Some examples are in Figure 11.

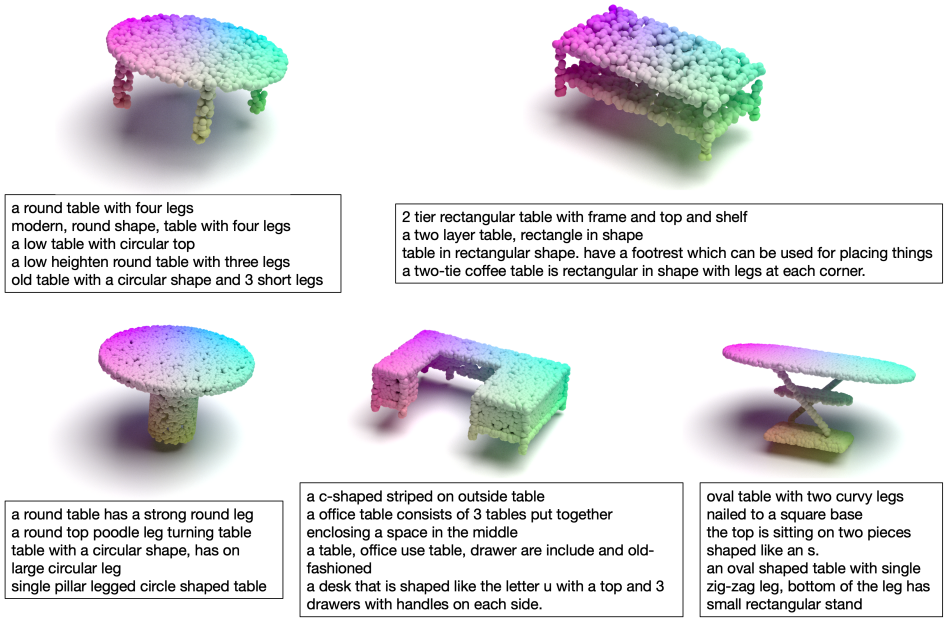


Figure 11: Random examples in adjusted Text2Shape dataset. One description per sentence.

ShapeGlot: (Achlioptas et al., 2019) provides many meaningful structural descriptions for chairs. However, the descriptions are about triple relationships. It provided users with two shapes and a comparative description and asked users to choose which shape most satisfied the description, i.e. (*Shape A*, *Shape B*, *Text*). We transfer the triples into doubles by assigning the text to the target shape according to annotations. However, some texts lose their meanings if they do not have the control shape. Furthermore, we found single text in ShapeGlot is usually not very informative that can plot the target shape accurately. In the end, we filter out the less-meaningful texts and concatenate several shapes' texts together as one description. Also, we remove superlative words, like longest, because they usually cannot hold across the entire dataset. For a few data which lack structural descriptions, we will annotate them manually. Some examples are in Figure 12.

ABO: (Collins et al., 2021) provides high-quality CAD models and descriptions in different languages for amazon products. We only use the description of the 'item_name' under the language tag 'en_US'. To add structure-related texts to ABO shapes, we collect more descriptions by manually annotating. Also, since ABO consists of amazon products, some original descriptions contain brand names, such as "amazon basics" and "ravenna", which are not connected to shape geometry. Also, some

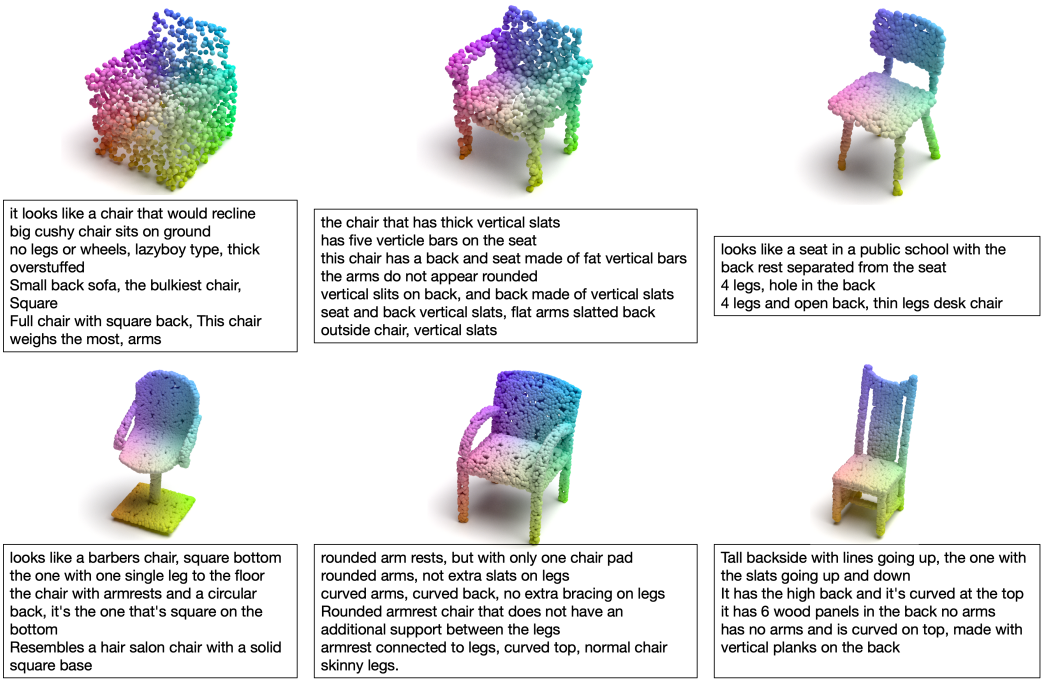


Figure 12: Random examples in adjusted ShapeGlot dataset. One description per sentence.

descriptions contain product dimensions like $16.0 \times 8.2 \times 7.4$. Since we normalize all point clouds into a unit ball, those dimensions are not connected to its scale. Therefore, we deleted the descriptions related to brand names and dimensions in the descriptions. Some examples are in Figure 13.

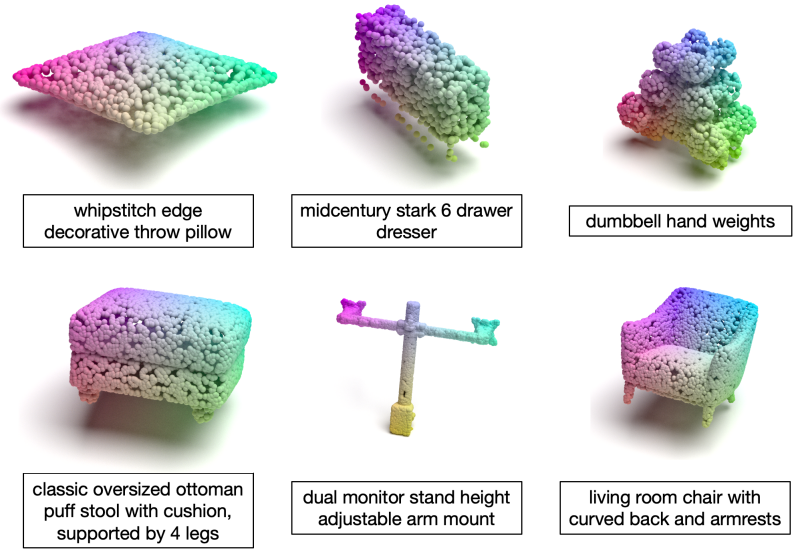


Figure 13: Random examples in adjusted ABO dataset. One description per sentence.

D.3 PARITALSHAPE-COMPLETESHAPE PAIRS

We used the data from GenRe (Zhang et al., 2018) which has 20 random view renderings of airplanes, cars, and chairs from ShapeNet, for a total of $20 \times 9{,}646$ training (*point-clouds*, *point-clouds*) pairs and $20 \times 1{,}382$ test (*point-clouds*, *point-clouds*) pairs. We sample 200 points from the above depth images as inputs, sample 2048 points from the corresponding 3D shapes as targets, and evaluate over

all the provided 20 views. Unlike previous benchmarks (Zhou et al., 2021), which train a model for a single shape category, we train every completion model with all training data.

E TRAINING & MODEL DETAILS

This section provides the training and model details for our *Point*VQVAE and ShapeCode Transformer used in our main paper, and the *Program*VQVAE we mentioned in Section 3.1. We implement and train our models with PyTorch (Paszke et al., 2019) and 8 A40 GPUs.

E.1 *Point*VQVAE

For *Point*VQVAE, we adopt the cosine annealing strategy of SGDR (Loshchilov & Hutter, 2016) (w/o restarts) as our optimization scheduler and set the initial learning rate 0.002 with 8×32 batch size and 200 epochs.

*Point*VQVAE encoder has two hierarchical layers ($\#T = 2$). In the first layer, we sample 512 center points. Then, we gather information from 64 points within $R_{T_1} = 0.1$ ball around each center point with two fully connected layers [64, 512] and [512, 512]. After the first layer, our point cloud size will be reduced to 512 from 10,000 (original input size). Similarly, in the second layer, we sample 128 center points, and do $R_{T_2} = 0.4$ ball query with 512 points. We also adopt two fully connected layers here with shapes [512, 512]. After the encoding phase, we will have 128 embeddings and look for the closest embedding in the codebook that has the size of [512, 512]. We concatenated the indexed embedding to form a [128, 512] embedding and sent it to the decoder. The structure of *Point*VQVAE decoder is shown in Figure 2, we have batch norm 1D (Ioffe & Szegedy, 2015) and ReLU (He et al., 2015) following each convolutional layer. The output of the residual-like structure (Figure 2) has the shape [#Batch Size, 512, 128]. We then applied a max-pool layer to the last dim of the output to obtain an embedding with shape [#Batch Size, 512, 1]. The max-pool layer can eliminate the effect caused by the concatenation order of the codebook embedding, similar to the use of max-pool layers in PointNet (Qi et al., 2017a). After the max-pool layer, we finally have two Convolutional 1D layers with [512, 1024] and [1024, 2048 * 3] to predict 2,048 points. Batch norm 1D and ReLU are also used between the last two layers.

We use Chamfer Distance (CD) and Earth Mover's Distance (EMD) as the reconstruction loss to serve as one of the objectives of *Point*VQVAE. Specifically, we adopt the implementation of CD in Pytorch3D (Ravi et al., 2020) and implemented an EMD approximation via auction algorithm (Bertsekas & Castanon, 1989) with the help of codes provided in (Fan et al., 2017; Liu et al., 2020b). The EMD codes will be released with a detailed report.

$$\mathcal{D}_{EMD}(P_1, P_2) = \min_{\phi: P_1 \to P_2} \sum_{x \in P_1} \|x - \phi(x)\|_2, \quad \phi \text{ is a bijection} \tag{3}$$

$$\mathcal{D}_{CD}(P_1, P_2) = \sum_{x \in P_1} \min_{y \in P_2} \|x - y\|_2^2 + \sum_{y \in P_2} \min_{x \in P_1} \|x - y\|_2^2 \tag{4}$$

$$\tag{5}$$

The overall objective for optimizing *Point*VQVAE adopted the one used in (Van Den Oord et al., 2017) and is shown below, where sg represents stop gradient operations, $\mathcal{Z}_e(p)$ refers to the output of our encoder, e is the nearest embedding in our codebook with the encoder output. The first two terms are the reconstruction loss to push the final output of *Point*VQVAE closer to input point clouds. The third term is $\mathcal{L}_2$ loss to move the embedding in our codebook closer to the corresponding encoder output. The fourth one is the commitment loss to constrain those embedding growing.

$$\mathcal{L}_{total} = \mathcal{D}_{EMD} + \mathcal{D}_{CD} + \|\text{sg}[\mathcal{Z}_e(p)] - e\|_2^2 + \|\mathcal{Z}_e(p) - \text{sg}[e]\|_2^2$$

E.2 SHAPECODE TRANSFORMER

We also adopt the cosine annealing strategy as our optimization scheduler and set the initial learning rate 0.001 with 8×24 batch size and 100 epochs. For generality, we adopt the simplest Transformer

(Vaswani et al., 2017) with depth 5, wide 64, and 8 attention heads. We integrate full attention masks (Child et al., 2019) to force it to do autoregressively predictions (Van den Oord et al., 2016).

As mentioned in Section 3.2, we have different positional embedding for each type of data pair (i.e., task), which will be optimized during the training. Since we pad 0 at the leftmost of our data, the positional embedding for the conditional code will have an extra token. For example, if we conduct (*Text*, *Point Cloud*) transformation, we will have $P_{text} = 256 + 1 = 257$ positional tokens for Text and $P_{point} = 128$ for Point Cloud. Therefore, we have a total of $257 + 128 = 385$ positional tokens for (*Text*, *Point Cloud*). For each positional embedding, we have the embedding dimensional 512. As a result, we have embedding size $[385, 512]$ for the task (*Text*, *Point Cloud*). For each task, Neural Shape Compiler has different positional embeddings but the same process described above.

Also, ShapeCode Transformer has a self-maintained embedding matrix for each type of code, as illustrated in Section 3.2. We also use (*Text*, *Point Cloud*) transformation as example, where Text code has $N_{text} = 256$ tokens and Point Cloud code has $N_{point} = 128$. The embedding dimension is 512. Therefore, we have embedding matrix with size [N_{text}, 512], [N_{point}, 512], and [$N_{program}$, 512] for Text, Point Cloud, and Program, respectively.

The overall objective of ShapeCode Transformer is the summation of loss computing over each pair of data: (1) (*Text*, *Point Cloud*); (2) (*Point Cloud*, *Text*); (3) (*Point Cloud*, *ShapeProgram*); (4) (*Partial PointCloud*, *Complete PointCloud*). For each pair, we define the loss over every token with Cross-Entropy loss as shown below, where $\mathcal{L}$ indicates cross-entropy. Noting that, for program codes $\mathcal{C}^{program}$, we will turn negative integers into positive integers, like mapping [-20, 20] to [0, 40].

$$\sum_{k} \mathcal{L}(c_k, c_k^{gt} | [0, \cdots, c_{k-1}])$$

E.3 *Program*VQVAE

Currently, Shape Programs (Tian et al., 2019) only has limited discrete parameters, and thus, we adopted the way of discretizing parameters into codes in our main paper. For example, we will map integers in [-20, 20] to [0, 40] to assign unique positive integers for each parameter. Although the current shape program can already cover a lot of 3D shapes, it may be upgraded to encode more detailed structures of 3D shapes that need to involve continuous parameters. For example, the statement *draw('Top', 'Square', P=(-1,0,0), G=(2,5))* in Figure 1 may have decimal parameters *draw('Top', 'Square', P=(-1.2,0,0.3), G=(2,5.7))*. Furthermore, the primitive basis used in the shape program may be replaced with a functional basis (Mescheder et al., 2019; Genova et al., 2020). As mentioned in Section 3.1, we also designed *Program*VQVAE to transform shape programs into codes to fill the gap in handling continuous parameters.

The design of *Program*VQVAE is pretty similar to our *Point*VQVAE, where we leverage the most basic concepts of vector quantization and adopt the straight-through gradient estimator and the vanilla codebook objective used in (Van Den Oord et al., 2017) for optimizing. Given a statement, we will turn its statement type into a one-hot vector and concatenate it with its parameters as the input to *Program*VQVAE. For example, we will turn *draw('Top', 'Square', P=(-1,0,0), G=(2,5))* into $[3, -1, 0, 0, 2, 5, 0, 0]$ where the value 3 is the represents the statement type 'draw'. The encoder and decoder will be two vanilla Transformer (Vaswani et al., 2017) with depth 3 and MLP dimension 32, and the codebook is [512, 512], the same size as we used in *Point*VQVAE. We use cross-entropy to compute the loss for predicting statement types and $\mathcal{L}_2$ loss for regressing statement parameters.

We synthesize 200,000 (*Point Cloud*, *Program*) pairs for training the *Program*VQVAE. On 10,000 validation data, it achieved 100% accuracy in predicting statement types and < 0.01 L2 distance in parameter regression. The results show the proposed *Program*VQVAE can well learn the distribution of our synthesized shape programs.

E.4 Baselines Implementation Details

This paper compared various baselines across different tasks to examine the performance of the proposed framework. We generally follow their provided codes and default configurations for each

baseline, including training epochs, learning rates, and other hyper-parameters. We only modify the code when changes are needed to use our dataset. Some modifications are listed in this section.

- Shape IMLE (Liu et al., 2022b) preprocesses all texts with Bert (Devlin et al., 2018) in its framework. We adopted the same way and used *BertTokenizer* of the pre-trained flag "bert-base-uncased" to process all the texts in our dataset.
- We adopted *clip.tokenize* and *clip.encode_text* to encode all test texts and input them into CLIP-Forge (Sanghi et al., 2022) for quantitative comparisons. We input the text prompt to DreamField (Jain et al., 2021) by the provided command line in Gihub of Jain et al. (2021) for qualitative comparisons. In our environment, we need 4 GPUs of 2048 Gb memory to run a single text input with DreamField (Jain et al., 2021). It takes more than eight days to finish the default 10,000 iterations for single text input. Therefore, it is impractical to test DreamField in our test set, which consists of 15,000 text-shape pairs.
- CWGAN (Chen et al., 2018) provided processed data as templates. We follow their data formats to process our data and update each entry, including *caption_tuples*, *caption_matches*, *vocab_size*, *max_caption_length*, and *dataset_size*. We adopted spaCy (Honnibal et al., 2020) to tokenize texts as mentioned in Chen et al. (2018). (Chen et al., 2018) simply set the token number in a linear increase rule based on their data order, while we set our token number based on our data order. We also disregarded descriptions with more than 96 tokens as (Chen et al., 2018) did.
- For AutoSDF (Mittal et al., 2022), their current codebase did not support training generation models conditioned in texts; we do not compare it for now. We will try to add their results once the code is updated.
- For the point cloud completion baseline PVD (Zhou et al., 2021), we jointly train their models with three data classes. This differs from their original paper, which trains separate models for each shape class.

F *Point*VQVAE Discussions

F.1 Analysis of Learned Codebook

This section provides some preliminary analysis of the learned codebook of *Point*VQVAE. The codebook is the interface to connect the output of *Point*VQVAE encoder and the input of *Point*VQVAE decoder. In our experiments, the codebook size is [512, 512], and we have 128 embedding for a single input point cloud.

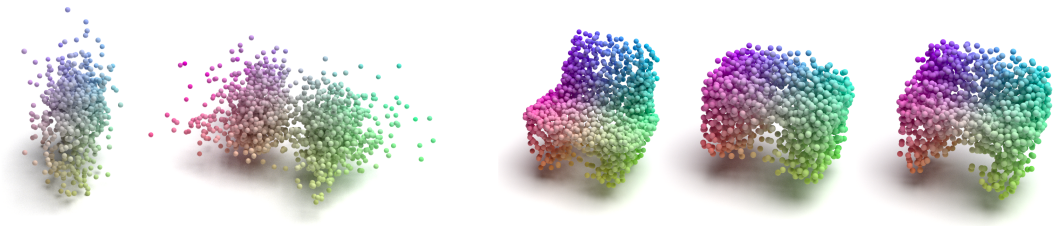


Figure 14: The right three shapes are reconstruction results of random codes and the left two shapes are reconstruction results of indexed codes illustrated in the text.

For each input point cloud, we will have 128 codes to index the embedding and feed it into the decoder later. We look at how many different embeddings of the codebook will be used for a single input point cloud. We compute the number with whole ShapeNet (Chang et al., 2015) shapes, where we auto-encode shape and compute the average ratio of unique codes. The result is 44.28, which indicates point cloud code $\mathcal{C}^{point}$ will have multiple replicate code numbers. Does this indicate that the codebook learns the basic primitives (Deprelle et al., 2019), selects the suitable ones for a specific point cloud, then fuses them for reconstruction? If so, will those primitives be perceptually understandable to our humans? To study this, we design indexed codes with all one number $[x, x, \cdots, x]$, like $[1, 1, \cdots, 1]$, length = 128. Those codes visualize the codebook's $\#x$ embedding. We feed the special codes into our decoder to reconstruct the corresponding embedding

and show two examples at the left of Figure 14. Results show the embedding is pretty random to humans and may exist in certain symmetry structures. Since the reconstruction results are highly correlated to the decoder weights, the way we adopted above may not be the ideal solution. We will try to develop other methods to understand how the 3D codebook works in the future. Besides, we also tried to randomly generate the codes and reconstruct them to see if the random samples from the latent discrete space are meaningful. Results are shown in the right of Figure 14, where they are somewhat reasonable in the structure. We also show an example in Figure 15 where we interpolate two different shape codes and reconstruct the intermediate results.

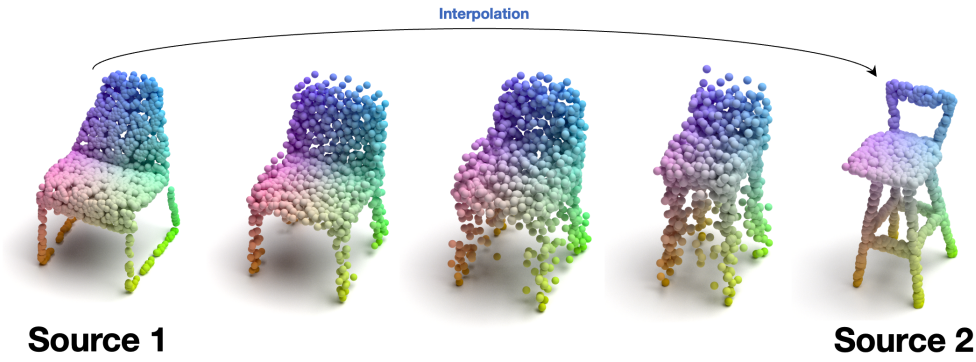


Figure 15: A random example of interpolating latent codes between two resource shapes of our proposed *Point*VQVAE.

F.2 Empirical Studies

In this section, we conduct empirical study about *Point*VQVAE, include ablation studies and comparisons. We changed the number of points in the last layer of the encoder (the color points in Figure 2) from 1 to 128. When the point number equals 1, our encoder equals the standard encoder of PointNet++ (Qi et al., 2017b) which uses a global pooling layer. Table 6 shows such a design brings a catastrophic performance drop due to it being significantly difficult for the codebook to encode holistic point clouds instead of parts of the point clouds.

Variants	CD ↓
*Point*VQVAE globalpool	16.63
*Point*VQVAE final-16	10.26
*Point*VQVAE final-32	8.57
*Point*VQVAE final-64	7.34
*Point*VQVAE final-128	6.93
*Point*VQVAE final-256	6.78

Methods	CD ↓
l-GAN (Achlioptas et al., 2018)	9.23
Point Flow (Yang et al., 2019)	7.76
ShapeGF (Cai et al., 2020)	5.97
DiffGen (Luo & Hu, 2021)	5.62
3DCM (Cheng et al., 2022)	6.32
*Point*VQVAE Limited	6.93
*Point*VQVAE	5.98
Oracle	3.09

Table 6: Point Cloud auto-encoding performance on ShapeNet. CD is multiplied by 10^3. X in *Point*VQVAE final-X means how many points we sampled in the last layer of the encoder. Globalpool indicates only having one point with the global receptive field. All the models besides *Point*VQVAE are trained on ShapeNet (Chang et al., 2015) shapes. *Point*VQVAE is trained on all our 3D shapes.

Also, we compare with some point clouds based auto-encoders (Achlioptas et al., 2018; Yang et al., 2019; Luo & Hu, 2021; Cheng et al., 2022). For each shape, we sample 2048 points thereon and use Chamfer Distance (CD) to measure the difference between inputs and reconstruction results. We will report Chamfer Distance $\mathcal{D}_{CD}(P_1, P_2)$ as the summation of both two-direction $P_1 -> P_2$ and $P_2 -> P_1$. EMD is not reported because each method has its implementation of EMD, and none of the methods can guarantee an approximation of EMD based on their GitHub code. In contrast, all the used CD implementations are correct. The results are shown in Table 6. The numbers under "Oracle" mean the lower bound of reconstruction, obtained by computing the distance between two different

samplings from the same shape meshes. Compared to *Point*VQVAE Limited, *Point*VQVAE achieved higher scores which means training over more 3D data can help *Point*VQVAE model the 3D shape distribution and also not reduce its performance on the part of the training data.

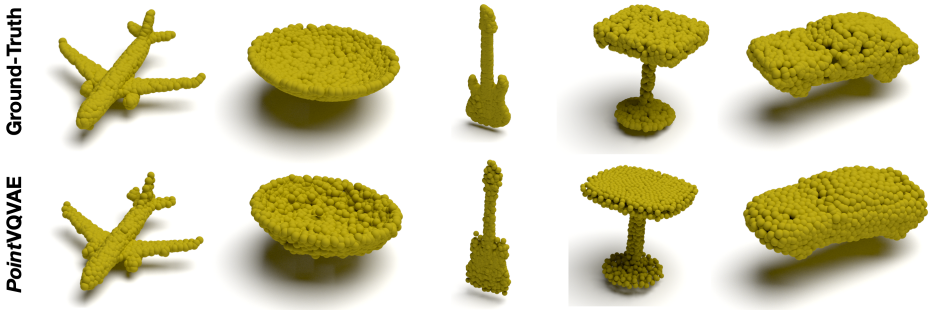


Figure 16: Random reconstruction examples of *Point*VQVAE.

G *Point Cloud* $\Longrightarrow$ *Program* DISCUSSIONS

In this section, we mainly discuss two points around *Point Cloud* $\Longrightarrow$ *Program*: (1) the generated sample numbers used in our shape program experiments; (2) the guided adaptation used in (Tian et al., 2019).

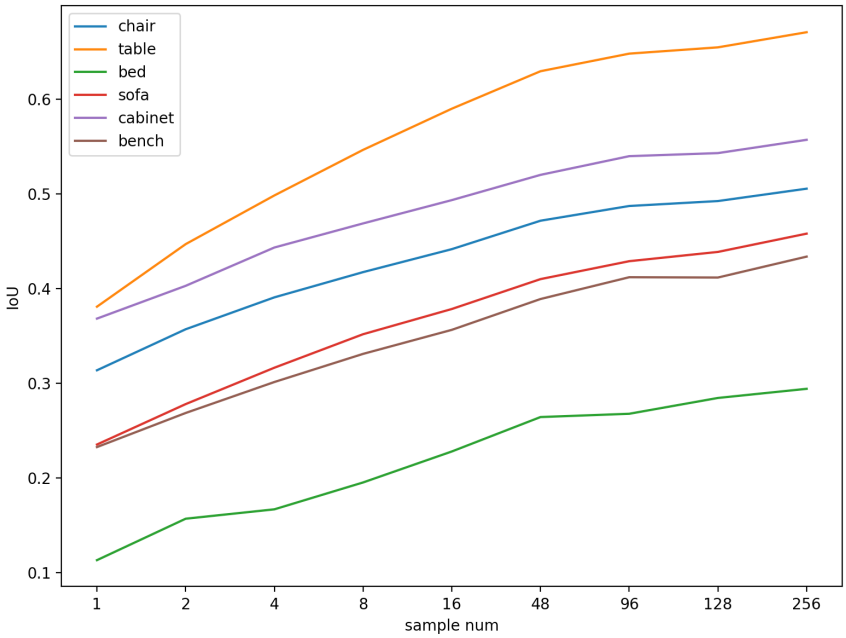


Figure 17: *Point Cloud* $\Longrightarrow$ *Program* performance versus the shape generated sample numbers.

We adopt 48 as the sample number in *Point Cloud* $\Longrightarrow$ *Program* experiments because all other tasks adopt batch size 48. Here, we provide a more thorough study of the sample numbers. We test Shape Compiler performance in all six shape categories with sample numbers [1, 2, 4, 8, 16, 48, 96, 128, 256]. The results are shown in Figure 17. The results show that the proposed method achieved higher performance by increasing the generated sample numbers. This is also one of the benefits of the generation framework we adopted in our approach.

Shape Program (Tian et al., 2019) used guided adaptation in finetuning their models to new test data. They achieved higher performance by using guided adaptation. However, in our experiments,

we intended not to follow (Tian et al., 2019) to perform guided adaptation. We choose this way because we aim to take a step further in shaping program generation for practical uses. In real-world applications, we usually cannot afford the extra training time for conducting "guided adaptation"; sometimes, we even do not have enough computation to run adaptation (e.g., edge computing). Also, generating shape programs without "guided adaptation" can enable us to use one class-agnostic model to process shapes from different categories instead of having class-specific models for each class. The proposed framework, Shape Compiler, only has one class-agnostic model for processing six different shape categories. However, Tian et al. (2019) will need to train six different models to process the test shape if they use guided adaptation. Combining the advantages of time-saving and class-agnostic, PointCloud-to-Program could take a step toward achieving real-world impact.

H MORE RESULTS

In this section, we provide more generated samples by the proposed Neural Shape Compiler. Extra results are contained in the supplementary.

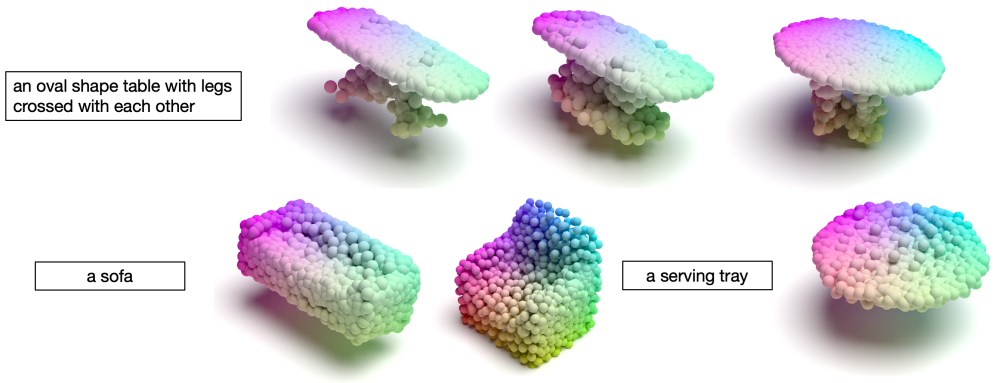


Figure 18: Generated shapes by Neural Shape Compiler given different text prompts. Shape Compiler can generate multiple shapes given a text query.

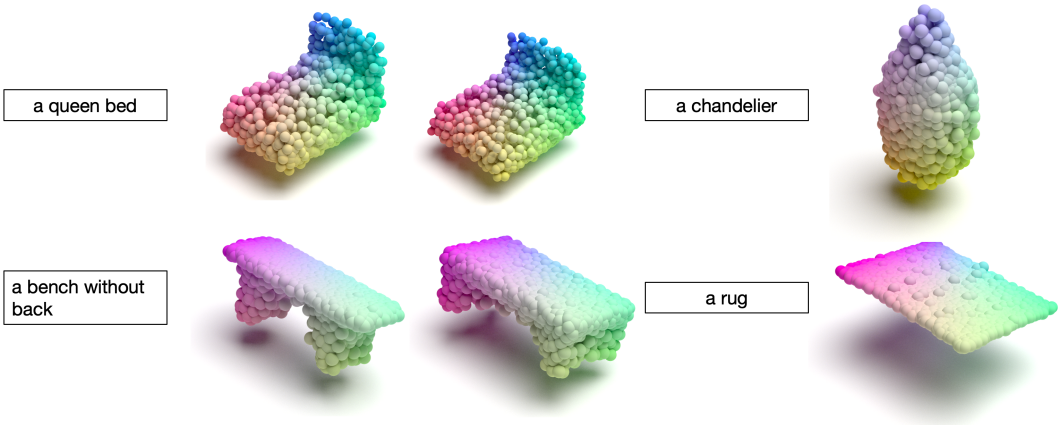


Figure 19: Generated shapes by Neural Shape Compiler given different text prompts. Shape Compiler can generate multiple shapes given a text query.

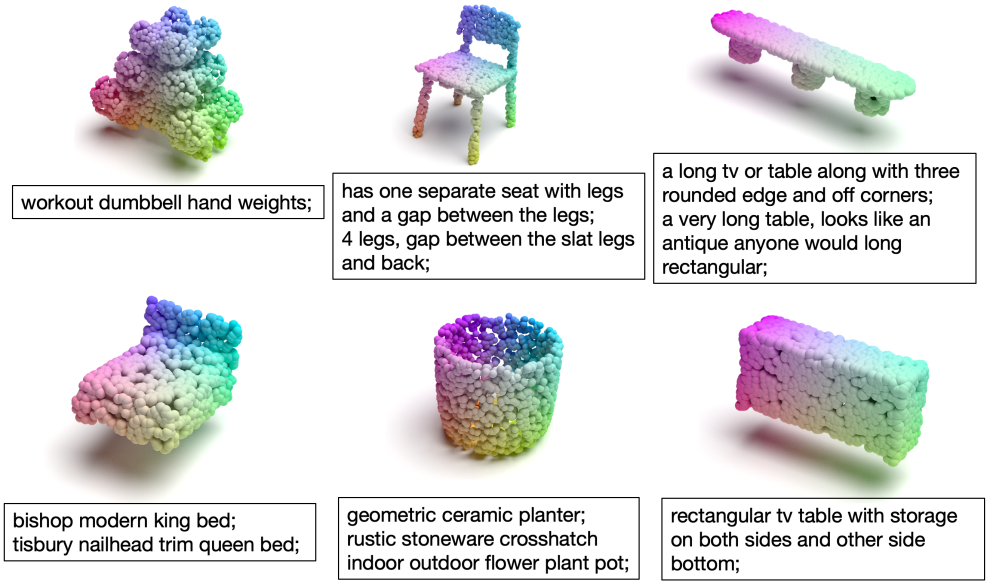


Figure 20: *Point Cloud* $\Longrightarrow$ *Text* results by Neural Shape Compiler given different point clouds. Shape Compiler can generate multiple captions given a point cloud query. One description per sentence.

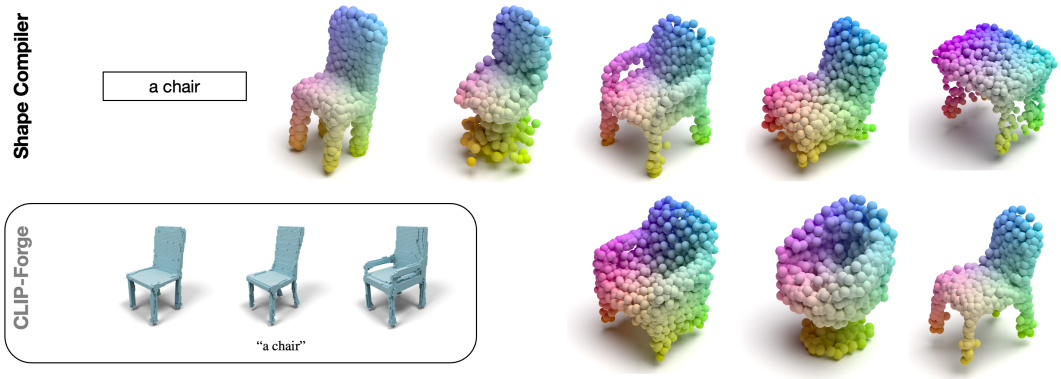


Figure 21: Directly comparing the performance of Neural Shape Compiler with CLIP-Forge (Sanghi et al., 2022) results on a simple text prompt.

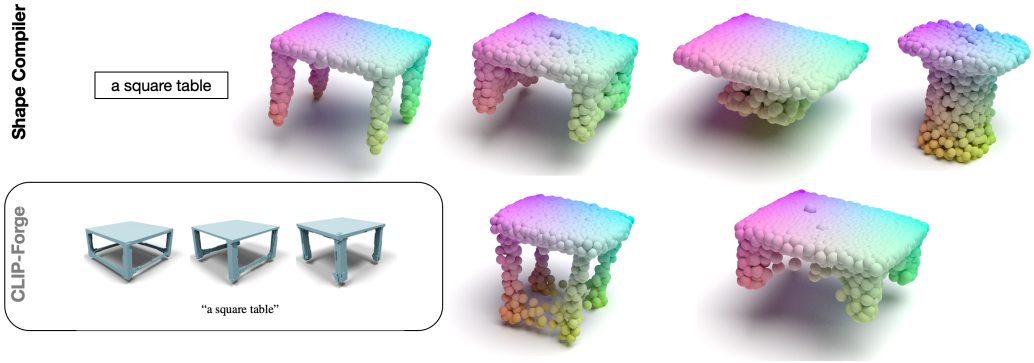


Figure 22: Directly comparing the performance of Neural Shape Compiler with CLIP-Forge (Sanghi et al., 2022) results on a simple text prompt.

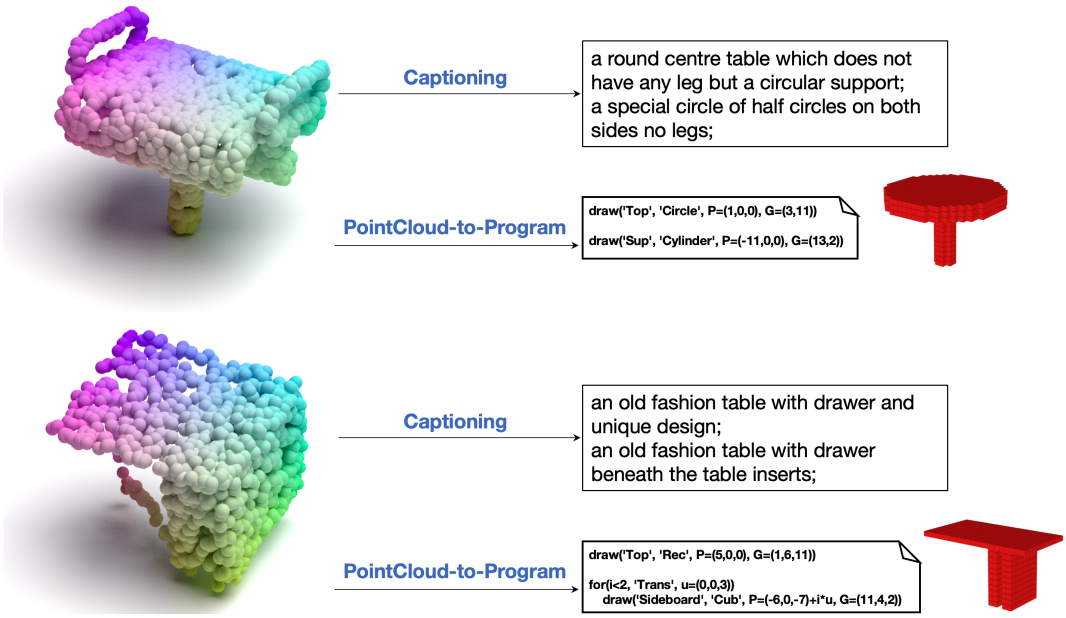


Figure 23: *Point Cloud* $\Longrightarrow$ *Text* and *Point Cloud* $\Longrightarrow$ *Program* results by Neural Shape Compiler given the input point clouds. One description per sentence.