ShapeWalk: Compositional Shape Editing through Language-Guided Chains

Habib Slim
KAUST
habib.slim@kaust.edu.sa

Mohamed Elhoseiny
KAUST
mohamed.elhoseiny@kaust.edu.sa

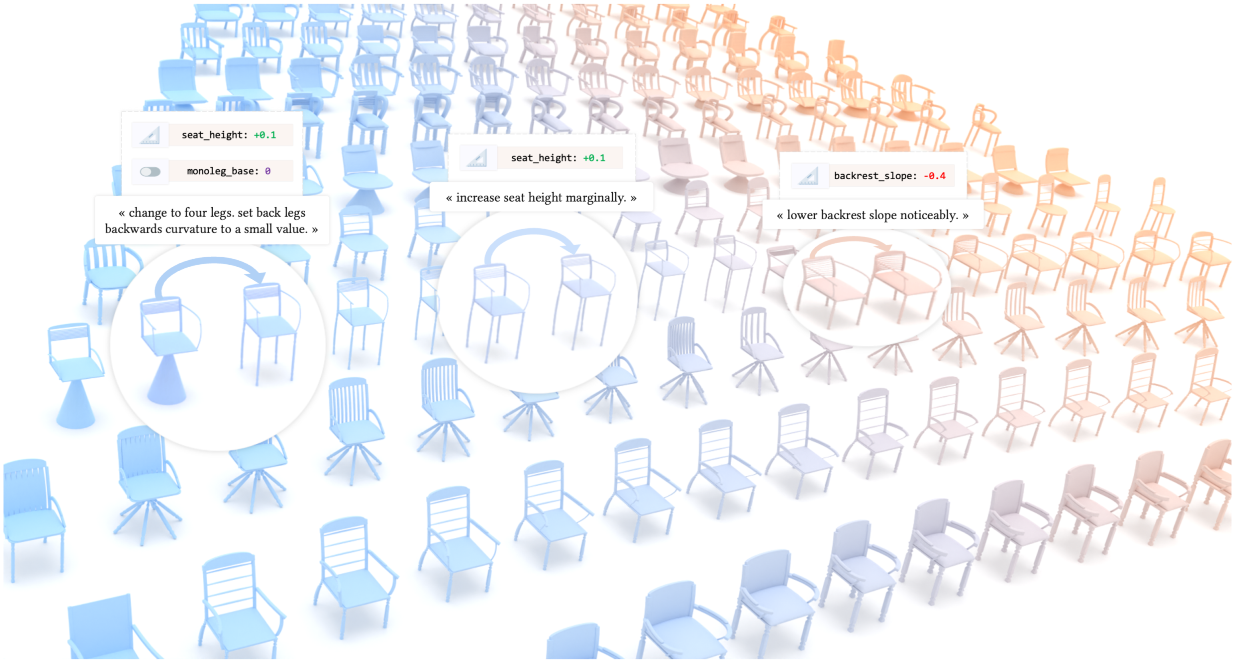

seat_height: +0.1
monoleg_base: 0
« change to four legs. set back legs backwards curvature to a small value. »
seat_height: +0.1
« increase seat height marginally. »
backrest_slope: -0.4
« lower backrest slope noticeably. »

Figure 1. **Visualizing shape chains from ShapeWalk.** Our dataset consists of 158K unique shapes connected through 26K edit chains, with an average length of 14 chained shapes. We illustrate the interpolation process by coloring shapes based on their proximity to the starting shape (**blue**), and the ending shape (**orange**). Each consecutive pair of shapes is associated with precise language instructions describing the applied edits. For each shape transition, we also provide a precise edit vector $\overline{\theta_{ij}}$ describing the parameter changes necessary to transition from one shape to the next.

Abstract

Editing 3D shapes through natural language instructions is a challenging task that requires the comprehension of both language semantics and fine-grained geometric details. To bridge this gap, we introduce ShapeWalk, a carefully designed synthetic dataset designed to advance the field of language-guided shape editing. The dataset consists of 158K unique shapes connected through 26K edit chains, with an average length of 14 chained shapes. Each consecutive pair of shapes is associated with precise language instructions describing the applied edits. We synthesize edit chains by reconstructing and interpolating shapes sampled from a realistic CAD-designed 3D dataset in the parameter space of the GeoCode shape program. We leverage rule-based methods and language models to generate accurate and realistic natural language prompts corresponding to each edit. To illustrate the practicality of our contribution, we train neural editor modules in the latent space of shape autoencoders, and demonstrate the ability of our dataset to enable a variety of language-guided shape edits. Finally, we introduce multi-step editing metrics to benchmark the capacity of our models to perform recursive shape edits. We hope that our work will enable further study of compositional language-guided shape editing, and finds application in 3D CAD design and interactive modeling.

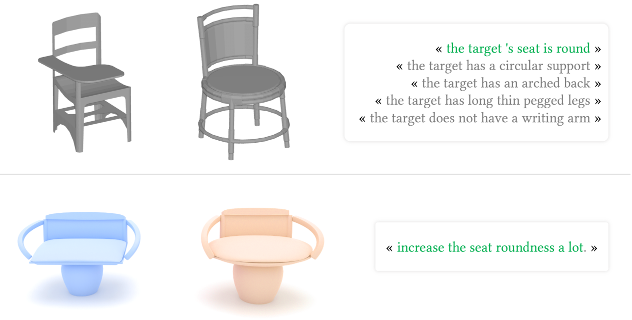

« the target 's seat is round »
« the target has a circular support »
« the target has an arched back »
« the target has long thin pegged legs »
« the target does not have a writing arm »
« increase the seat roundness a lot. »

Figure 2. **Comparing ShapeTalk with our work.** We compare the ShapeTalk [3] dataset (*top*) with our work (*bottom*). For an equivalent edit instruction (in green), ShapeTalk provides pairs of shapes with many factors of variation, while we generate synthetic pairs of shapes with a single clear varying factor.

1. Introduction

Whether in the realms of computer-aided design, virtual reality environments, or digital content creation, the process of refining and enhancing 3D visual data often involves intricate adjustments to geometric shapes, textures, and lighting. Furthermore, the necessity for precise modifications adds an additional layer of complexity, as inaccuracies can have profound implications on the final output. This labor-intensive nature of 3D data editing not only hinders workflow efficiency but also poses a barrier for individuals without specialized skills, limiting the democratization of 3D content manipulation. As such, there exists a compelling need for innovative solutions that streamline and democratize the 3D editing process, making it more accessible to a broader range of users. But how to train models with the ability to perform these complex edits?

To that end, we introduce ShapeWalk, a dataset aiming at addressing the challenges inherent in the intricate and skill-intensive nature of 3D data editing. Our dataset consists of synthesized chains of 3D shapes, connected through edit vectors associated with precise language instructions describing the applied edits. It contains 158K unique shapes connected through 26K edit chains, with an average length of 14 chained shapes. Our generation method is scalable and can yield an indefinite number of realistic shape chains, and extended to any 3D domain and shape program. To our knowledge, our dataset is the first to provide a large-scale collection of realistic 3D shape edit chains with precise language instructions, and matching ground-truth edited shapes. Our method yields specific edits undiluted by other factors of variation, as illustrated in Figure 2.

To validate the usefulness of our dataset, we train neural editor modules in the latent space of shape autoencoders [1], and demonstrate the ability of our dataset to train models capable of performing a variety of shape edits. We learn latent mappings in the space of a frozen shape autoencoder, and show that our method can be applied on shapes unseen during the training of the neural editor. Taking advantage of the ground-truth shape chains provided by our dataset, we introduce multi-step editing metrics inspired from the field of trajectory prediction [4, 26] to evaluate the quality of our models.

Our contributions can be summarized as follows:

- **Chained Shape-Editing Dataset:** We introduce a synthetic dataset of chained edits, with associated language instructions and ground-truth edited shapes. This is the first dataset of its kind, and is designed to facilitate the study of compositional shape editing.
- **Neural Shape Editor:** We introduce a neural editor module trained on our dataset, and demonstrate its ability to perform a variety of shape edits. We show that our method can be applied on shapes unseen during the training of the neural editor.
- **Chained-Editing Metrics:** With the introduction of ground-truth shape chains, we propose multi-step editing metrics to evaluate the ability of neural editors to perform recursively applied edits.

2. Related work

3D Shape Programs. A significant body of work [9, 11, 12, 14, 19] is dedicated to exploring the use of procedural programs for 3D shape representation. Representing 3D objects as visual programs has many advantages. Representing shapes as visual primitives enhances interpretability [11, 14], as programs are by definition human-readable and thus understandable by human experts. Shape programs are more compact [9, 27] than their usual 3D shape modalities, and can be used to represent shapes more efficiently. Programs can also be composed to create shapes [11, 14], or decomposed into smaller programs. Ideally, shape programs can be edited [13] to modify the underlying geometry they represent.

Our dataset consists of a collection of synthetic shapes created utilizing a backbone mesh-generating shape program. Specifically, we build from GeoCode [17], a 3D shape synthesis technique addressing the challenge of mapping high-fidelity geometry to an editable parameter space. GeoCode introduces a procedural program enabling the generation of high-quality mesh outputs with a balanced blend of interpretability and fine control.

Language-Guided 3D Shape Editing. Various works have explored the use of natural language instructions to guide 3D shape editing. Recent efforts [16, 20, 21] propose to leverage pre-trained CLIP [24] models to align

3D shapes with a given text prompt. Early efforts in that direction [16] included optimizing meshes to progressively align them with CLIP embeddings, leading to important computational overhead. CLIP-Sculptor [21], a more recent work, leverages a voxelized representation and a discrete latent space conditioned on CLIP's image-text embeddings to perform fast and fidel shape edits without shape optimization.

Shape Editing Datasets. Other works leverage large text-aligned 3D shape datasets and require significant manual annotation effort. ShapeCrafter [8] generates 3D shapes incrementally from text using a neural network, evolving with additional phrases. ShapeCrafter is designed for recursive shape edition and utilizes a VQ-VAE [23] model to represent shapes as discrete codes. This method exhibits consistent shape-text alignment with gradual evolution. ChangeIt3D [3] introduces ShapeTalk, a large dataset for describing 3D shape differences. The framework facilitates language-based editing of 3D models without requiring 2D to 3D conversion methods, and learns a shape editing model by learning contrasts between sampled shape pairs.

ShapeTalk [3] is the most similar work to our proposed dataset, and is the only publicly available dataset specifically designed for language-guided 3D shape-to-shape editing. ShapeTalk is a remarkable contribution in the field of language-guided shape editing, and is one of the first works to leverage large-scale 3D shape datasets to facilitate the study of language-guided shape editing. However, this work has some limitations that we attempt to address in our work. Collecting a dataset of shape differences is a challenging task requiring considerable manual annotation effort, and gathering a large number of 3D shapes may not be feasible for all domains. Furthermore, ShapeTalk is composed of edit contexts (i.e. shape pairs and edit instructions) which are hard to separate into fine-grained, composable edits. Most of the time, singular shape edits do not have have an exact ground-truth in the form of a source and target shape pair. We illustrate this important distinction in Figure 2. The availability of singular shape edits with an associated ground-truth would ease benchmarking the quality of shape editing methods, and facilitate the training process of editing models. ShapeTalk also does not provide a mechanism to generate edit chains, which are necessary to study compositional shape editing. Finally, ShapeTalk does not easily enable the study of shape editing in a compositional manner. In contrast, our dataset is **1)** synthesized by augmenting a small set of diverse shapes and can easily be scaled up, **2)** composed of fine-grained edits and coarse-grained edits with exact ground-truths, and **3)** separated into edit chains, which are designed to facilitate the study of compositional shape editing.

3. ShapeWalk

Our dataset contains 158K shapes split into a *random* and *realistic* set. With each edge connecting two consecutive shapes, we also produce a text instruction generated using the parameter changes necessary to transition from one shape to the next. We detail the generation method of our dataset here, and summarize the process in Figure 3.

3.1. Dataset

Definition. Our dataset can be defined as a collection of directed graph paths (dipaths), denoted as $\mathcal{P}^{(k)} = (S^{(k)}, E^{(k)}, f^{(k)})$. For each shape chain $\mathcal{P}^{(k)}$ of length l, we denote:

- $S^{(k)} = \{s_{\theta_\mathbf{1}}^{(k)}, \ldots, s_{\theta_\mathbf{N}}^{(k)}\}$, the set of distinct shape nodes composing the chain. Each shape is defined by a set of parameters $\theta_i \subseteq \Theta$, where Θ is the linear parameter space of our shape program.

- $E^{(k)} \subseteq \{(s_{\theta_\mathrm{i}}^{(k)}, s_{\theta_\mathrm{j}}^{(k)}) \mid s_{\theta_\mathrm{i}}^{(k)}, s_{\theta_\mathrm{j}}^{(k)} \in V^{(k)},\ j = i + 1\}$, the set of edges linking each consecutive pair of shapes.

- $f^{(k)} : E^{(k)} \mapsto \{(\overline{\theta_{ij}},\ \mathbf{p_{ij}}) \mid \overline{\theta_{ij}} \subseteq \Theta,\ \mathbf{p_{ij}} \in \Sigma\}$, a function mapping each edge to a vector $\overline{\theta_{ij}} \subseteq \Theta$ and a set of text instructions $\mathbf{p_{ij}}$. $\overline{\theta_{ij}}$ defines the parameter changes, or edits necessary, to go from shape i to j. $\mathbf{p_{ij}}$ is a set of text instructions describing this edit in natural language.

Generation. To generate our dataset, we start by reconstructing a set $S \subseteq \mathcal{S}$ of realistic 3D CAD shapes into the space of shapes covered by our shape program $\mathcal{S}_\Theta \subset \mathcal{S}$. To that end, we utilize shapes from the 3DCoMPaT^{++} [15, 22] dataset, a realistic, industry-based 3D CAD dataset. We employ this dataset to avoid overlap with ShapeNet [5], and to ensure diversity and visual quality of the shapes.
We define a visual similarity function $d : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}$ between two shapes $s_i, s_j \in S$ as the feature-wise mean squared error between the original and reconstructed meshes' renderings, in the space of a pre-trained ResNet-50 [10] encoder $\phi_R : \mathbb{R}^{h \times w \times 3} \mapsto \mathbb{R}^d$.

$$d(s_i, s_j) = \|\phi_R(f_R(s_i)) - \phi_R(f_R(s_j))\|_2$$

Where $f_R : S \mapsto \mathbb{R}^{3 \times h \times w}$ is a rendering function. The set of reconstructed shapes can then be defined as:

$$\underset{\substack{S_R \subseteq S \\ |S_R| = (1-\alpha)|S|}}{\arg\min} \sum_{s \in S_R} d(s, \widehat{s})$$

Where $\widehat{s}$ is the reconstructed shape, defined as:

$$\widehat{s} = f_\Theta \circ \phi_\Theta(s)$$

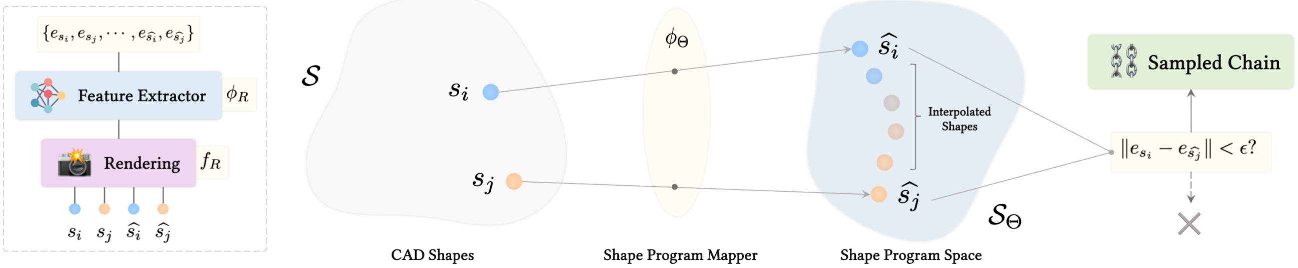


Figure 3. **Detailling the shape generation pipeline of ShapeWalk.** Realistic 3D CAD shapes are reconstructed from the 3DCoMPaT++ [22] dataset into the GeoCode [17] shape program parameter space, using a mapping function ϕ_Θ. Reconstructed shapes with an error over a fixed threshold are discarded using a visual similarity function d, which is based on rendered feature similarity. Filtered pairs of shapes are then interpolated in the parameter space of the shape program, to generate shape chains $\mathcal{P}^{(k)}$.

With $f_\Theta : \Theta \mapsto \mathcal{S}$ the shape program mapping shape parameters to a 3D mesh, and $\phi_\Theta : \mathcal{S} \mapsto \Theta$ an encoder based on DGCNN [17, 25] which samples points from a shape surface and regresses corresponding shape parameters in Θ.

We reconstruct shapes by first sampling pointclouds from the surface of the original meshes, and then feeding them into the DGCNN [25] encoder fine-tuned [17] to regress the parameters of the shape program. We discard a ratio of $\alpha = 0.08$ of the shapes with the highest reconstruction error, which we measure as the feature-wise mean squared error between the original and reconstructed meshes' renderings, in the space of a pre-trained ResNet-50 [10] encoder. This threshold is selected empirically by visually inspecting the shapes in the upper part of the reconstruction error distribution. In our dataset instance building on 3DCoMPaT++ shapes, we filter out a total of 89 shapes during this process, leading to a set of $|S_R| = 1113$ reconstructed shapes. This process can be scaled up to any dataset size and any shape domain, as long as a corresponding shape program is available.

Shape interpolation. After reconstructing a set S_R of realistic shapes, we generate a set of shape chains $\mathcal{P}^{(k)}$ by interpolating between the parameters of shape pairs.

To that end, we first sample a set of shape pairs $(s_{\theta_\mathbf{i}}, s_{\theta_\mathbf{j}}) \in \mathcal{S}_P \subseteq S_R \times S_R$, and partition them into $L = 10$ levels of proximity. These proximity levels are based on the feature-wise mean squared error between the original and reconstructed meshes' renderings, in the space of a pre-trained ResNet-50 [10] encoder.

We define the set of shape pairs $\mathcal{S}_P^{(l)}$ at level $l \in [\![1, L]\!]$ as:

$$\mathcal{S}_P^{(l)} = \{(s_{\theta_\mathbf{i}}, s_{\theta_\mathbf{j}}) \in \mathcal{S}_P \mid \overline{d(s_{\theta_\mathbf{i}}, s_{\theta_\mathbf{j}})} \in \left[\frac{l-1}{L}, \frac{l}{L}\right]\}$$

Where $\overline{d}$ is a visual similarity function, min-max normalized across all pairs of shapes.

For each pair $(s_{\theta_\mathbf{i}}, s_{\theta_\mathbf{j}}) \in \mathcal{S}_P^{(l)}$, we then interpolate between the shape parameters $\theta_\mathbf{i}$ and $\theta_\mathbf{j}$ to generate a set of intermediate shape parameters $(\theta_\mathbf{i}, \ldots, \theta_{\mathbf{i}+(\mathbf{N}-\mathbf{2})}, \theta_\mathbf{j}) \in \Theta^N$. The number of intermediate shapes N is defined as the number of differing parameters between $\theta_\mathbf{i}$ and $\theta_\mathbf{j}$, that is:

$$N = |\{k \mid \theta_\mathbf{i}[k] \neq \theta_\mathbf{j}[k]\}|$$

When the proximity level is low, edits necessary to transition from one shape to another are larger in intensity as the shapes are more different, and smaller when l is smaller. The ordering of intermediate edits is randomly sampled using a dependency-aware algorithm (for example, parameters relating to armrest height or width are sampled after the addition of armrests).

As illustrated in Figure 1, this generation process leads to plausible interpolated shapes, and to fine-grained and realistic sequences of edits with detailed metadata.

We summarize the process of generating realistic shape chains in Figure 3.

Random set. While the *realistic* set is based on reconstructed shapes from the 3DCoMPaT++ [15, 22] dataset, the *random* set is based on a collection of random shapes generated using the shape program. This alternative set aims at covering a large space of the parameter space of the GeoCode shape program. This subset is generated by systematically sampling various combinations of parameters within defined ranges determined by a minimum edit intensity, but does not necessarily result in realistic shapes.

3.2. Text Instructions

Rule-based generation. Given an edit vector $\overline{\theta_{ij}}$ describing the parameter changes necessary to transition from shape i to j, we generate a set of text instructions $\mathbf{p_{ij}}$ describing this edit in natural language. We first map each parameter $\theta_i[k]$ composing the edit vector to a natural language name, and map the magnitude of the

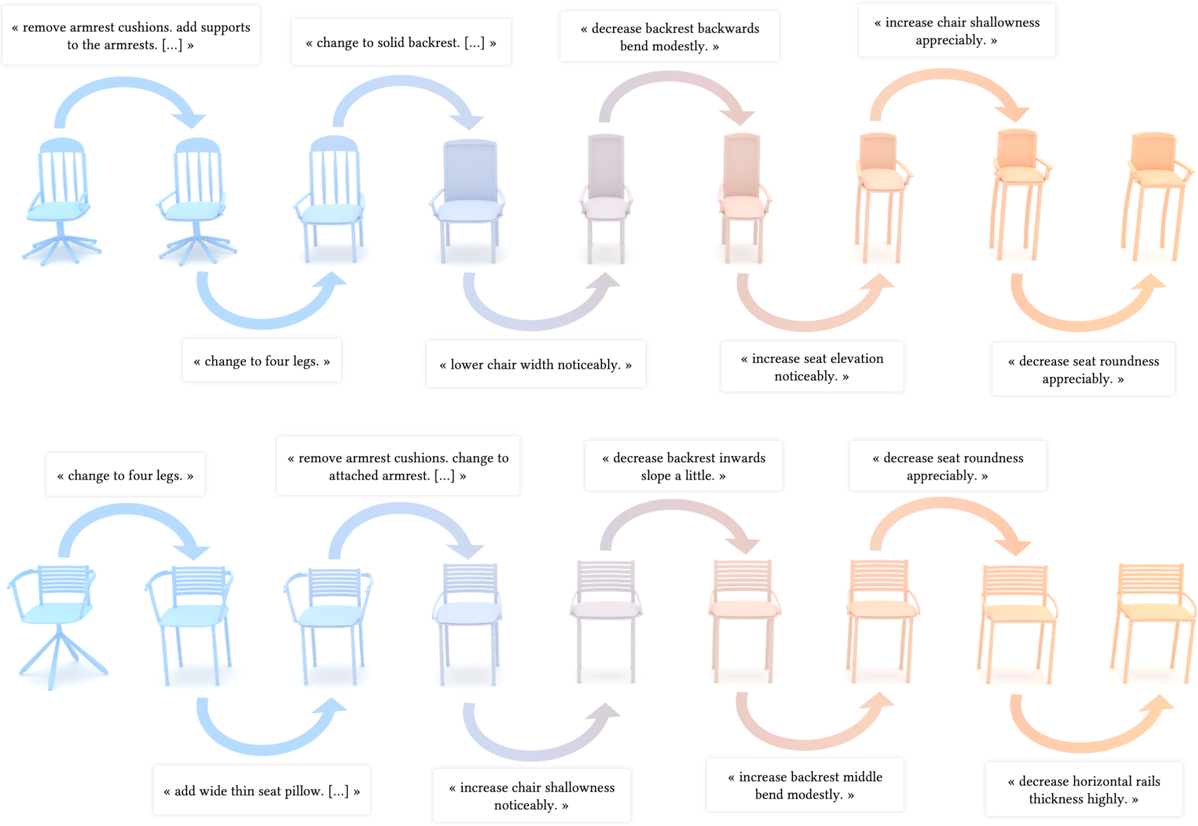

« remove armrest cushions. add supports to the armrests. [...] »
« change to solid backrest. [...] »
« decrease backrest backwards bend modestly. »
« increase chair shallowness appreciably. »
« change to four legs. »
« lower chair width noticeably. »
« increase seat elevation noticeably. »
« decrease seat roundness appreciably. »
« change to four legs. »
« remove armrest cushions. change to attached armrest. [...] »
« decrease backrest inwards slope a little. »
« decrease seat roundness appreciably. »
« add wide thin seat pillow. [...] »
« increase chair shallowness noticeably. »
« increase backrest middle bend modestly. »
« decrease horizontal rails thickness highly. »

Figure 4. **Dataset chains samples.** We feature in this figure shape chains from our dataset with associated generated text instructions. Each shape chain is composed of a sequence of shapes, with each consecutive pair of shapes associated to a set of text instructions describing the edits necessary to transition from one shape to the next. Overall, generated text instructions are accurate and provide a detailed description of the edits applied to the shapes. We include in the supplementary material additional samples of generated shape chains, alongside shape chains generated for two additional **table** and **vase** categories.

parameter change to a natural language intensity depending on the parameter type. Boolean parameters are also described appropriately, for example by using the word *add* or *remove* when describing the addition or removal of a part.

Augmentation. This rule-based instruction generation leads to perfectly accurate instructions, but may lack diversity. To alleviate this issue, we **1)** randomly sample adjectives and adverbs to characterize the intensity of edits, **2)** randomly invert parameter names and magnitude directions (e.g. "*increase armrests straightness*" will be augmented to "*decrease armrests bend*" for the same edit). As a final augmentation, we utilize a T5 [18] transformer model fine-tuned for paraphrasing to generate additional instructions for each edit vector. We use the Parrot library [6] to filter generated paraphrases based on a fluency and adequacy score.

We summarize the process of generating text instructions in the supplementary material.

4. Neural Shape Editing

4.1. Problem

We are concerned with the task of editing a shape s_i given a sequence of natural language edit instructions. We want to learn an editing function $f_E : \mathcal{S} \times \Sigma \mapsto \mathcal{S}$, where $\mathcal{S}$ is the space of input shapes, and Σ is the space of natural language edit instructions, able to compose edits from a starting shape s_1 to an ending shape s_N, given a sequence of edit instrutions $\{\mathbf{p_t}\}_{t=1}^{N}$.

$$\widehat{s_N} = f_E(\phi_T(\mathbf{p_{12}}),\, f_E(\phi_T(\mathbf{p_{23}}),\, \dots\, f_E(\phi_T(\mathbf{p_{kN}}),\, s_1)))$$

Where $N = |\mathcal{P}|$ is the length of the edit chain, and $\phi_T : \Sigma \mapsto \mathbb{R}^{D_T}$ is a text encoder. At the end of the edit sequence, we want to recover a shape $\widehat{s_N}$ as close as possible to the ground-truth shape s_N. Our proposed dataset provides a ground-truth for every intermediate shape s_k, which we leverage to train and evaluate our method.

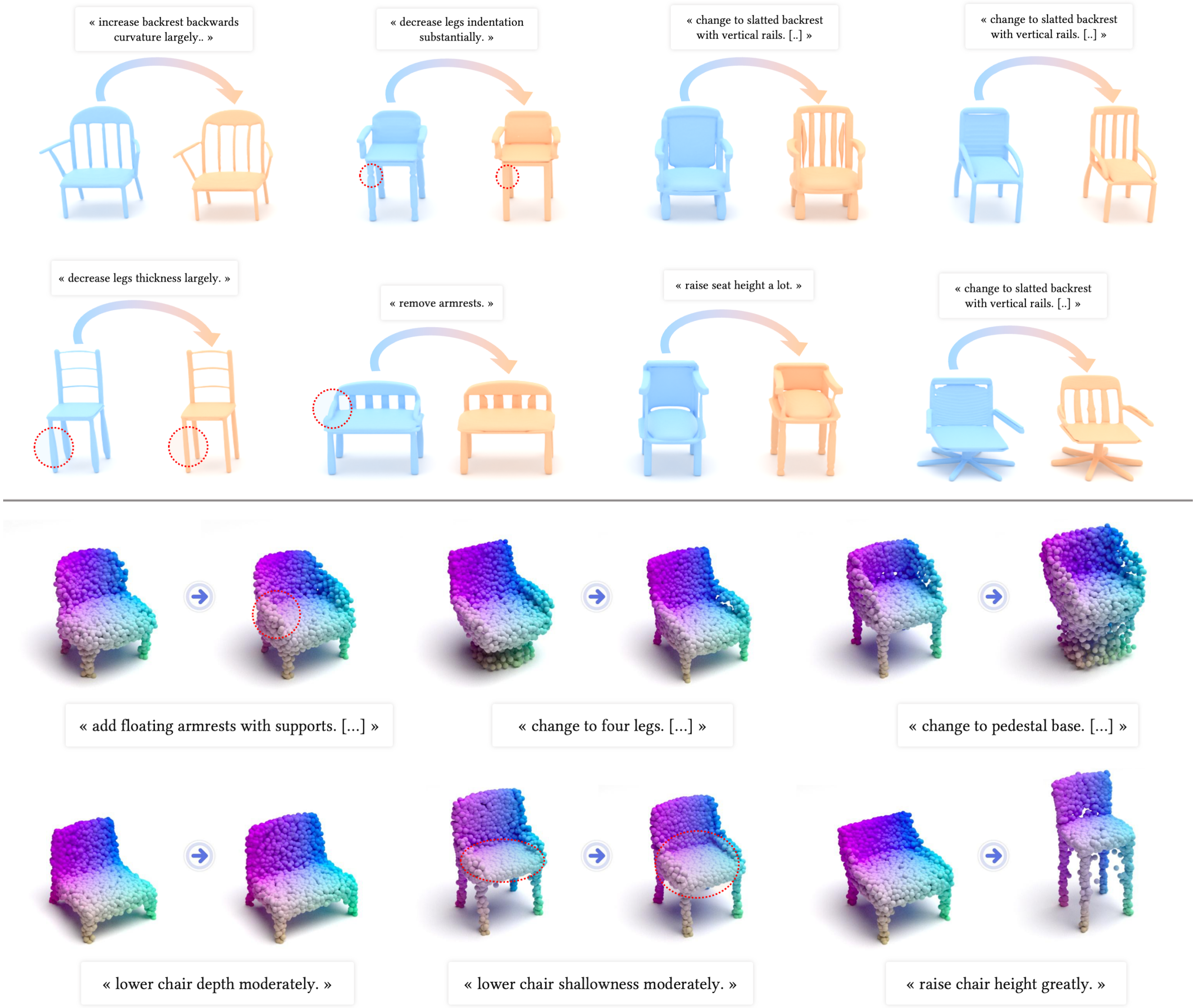

« increase backrest backwards curvature largely.. »
« decrease legs indentation substantially. »
« change to slatted backrest with vertical rails. [..] »
« change to slatted backrest with vertical rails. [..] »
« decrease legs thickness largely. »
« remove armrests. »
« raise seat height a lot. »
« change to slatted backrest with vertical rails. [..] »
« add floating armrests with supports. [...] »
« change to four legs. [...] »
« change to pedestal base. [...] »
« lower chair depth moderately. »
« lower chair shallowness moderately. »
« raise chair height greatly. »

Figure 5. **Visualizing synthesized pairs of edits using our latent editors.** We illustrate the diverse range of edits generated by our proposed latent mapper, showcasing its ability to capture nuanced variations in the input data, for 3DS2VS latents (top two rows) and PC-AE latents (bottom two rows). Each pair demonstrates the original input on the left and the corresponding synthesized output on the right, with the associated caption. We abbreviate instructions which are too long to fit in the figure, and indicate when text is omitted with an ellipsis. For more subtle changes, we circle the areas of interest in both source and target shapes. Overall, edits are consistent with the input instruction, preserve shape identity while generating plausible output shapes.

4.2. Objective

One of the main advantages of training a shape editing model with ShapeWalk compared to other works [2, 8] is the availability of exact ground-truth edited shapes for each edit instruction (see Section 2). We thus formulate the objective function of our learning problem as a simple $\mathcal{L}_2$ loss between the latents of the synthesized shape and the ground-truth shape. For all pairs of shapes (s_i, s_j) and their corresponding edit instructions $\mathbf{p_{ij}}$, we minimize the following loss:

$$\mathcal{L} = \| f_E(\phi_E(s_i),\ \mathbf{p_{ij}}) - \phi_E(s_j) \|_2$$

Where $\phi_{AE} = \phi_E \circ \phi_D$ is a shape autoencoder, and $\phi_E : \mathcal{S} \mapsto \mathbb{R}^{D_E}$ denotes its encoder component.

4.3. Models

For ϕ_{AE}, we experiment with PC-AE [1] and 3D2VS [28] models both pre-trained on ShapeNet [5]. Note that our method is agnostic to the choice of shape autoencoder and can be adapted to a variety of shape representations. For ϕ_T, we use a pretrained BERT [7] model.

Both ϕ_{AE} and ϕ_T are frozen during training, and we only train the parameters of our latent mapper f_E.

For PC-AE, we formulate our latent mapper as a neural module which for each tuple $(\phi_E(s_i),\ \mathbf{p_{ij}},\ \phi_E(s_j))$ predicts an edit vector $\widehat{\theta_{ij}} \in \mathbb{R}^{D_E}$ in the feature space of ϕ_E, and adds it to the latent of the input shape s_i to generate the latent of the output shape s_j:

$$f_E(\phi_E(s_i),\ \mathbf{p_{ij}}) = \widehat{\theta_{ij}} + \phi_E(s_i)$$

We utilize two variants of our latent mapper f_E: one in which the edit vector $\widehat{\theta_{ij}}$ is predicted directly, and one in which the edit vector $\widehat{\theta_{ij}}$ is predicted as a product of a normalized edit direction $\widehat{v_{ij}} \in \mathbb{R}^{D_E}$ and a magnitude $\widehat{m_{ij}} \in \mathbb{R}$ separately.

For 3D2VS which encodes shapes as latents sets of higher resolutions, we use a transformer-based latent diffusion model instead to generate edited latents. We concatenate the input shape latent $\phi_E(s_i)$ with the noised edited shape latent $\phi_E(s_j + \epsilon_t)$ and feed the BERT text embeddings to the cross-attention layers of the transformer blocks to predict the added noise. We illustrate our architecture for the latent mapper in the supplementary material.

5. Experiments

5.1. Chained Shape Editing

Comparison Models. For PC-AE, we experiment with three variants of our latent mapper module:

- DIRECTGEN directly predicts the edited shape latent without regressing an edit vector.
- LATEFUSION follows the network proposed in [3], and first passes the shape embeddings through an encoder. Encoded shape features are then concatenated with text embeddings and fed into a neural module which finally predicts the edit vector.
- OURS is a simple multi-layer perceptron (MLP) which directly takes the *context* (shape and text embeddings) as input to predict an edit vector.

For all of these variants, we experiment with various bottleneck dimensions and number of layers (in subscript).

We further decompose these variants into *coupled* and *decoupled* versions, where we predict a normalized edit direction $\widehat{v_{ij}} \in \mathbb{R}^{D_E}$ and a magnitude $\widehat{m_{ij}} \in \mathbb{R}$ separately. We provide in appendix the full architecture details and training hyperparameters employed to train our models.

Note that we do not compare with ChangeIt3D [2] as their neural-listener distillation method is not applicable in our context: we train our models with a direct feedback signal from the ground-truth edited shapes.

Single-step metrics. We propose to measure the quality of our editing steps both before and after shape reconstruction. Since we experiment on pointclouds, we use the scaled Chamfer Distance between the reconstructed original shape s_i and the reconstructed edited shape $\widehat{s_j}$:

$$d_{\text{CD}} = \frac{1}{|\widehat{s_j}|} \sum_{x \in \widehat{s_j}} \min_{y \in s_i} \|x - y\|_2 + \frac{1}{|s_i|} \sum_{y \in s_i} \min_{x \in \widehat{s_j}} \|x - y\|_2$$

We also use the $\mathcal{L}_2$ distance between the latents of the original shape s_i and the edited shape s_j:

$$d_{\mathcal{L}_2} = \|\phi_E(s_i) - \phi_E(s_j)\|_2$$

Multi-step metrics. To extend the single-step metrics to chained shape generation, we take inspiration from the trajectory prediction literature [4, 26] and propose appropriate metrics for our task.

Given a shape distance function $d : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}$, a chain of ground-truth reconstructed shapes $\{s_1, \ldots,\ s_N\}$, and a set of recursively generated shapes $\{\widehat{s_1}, \ldots,\ \widehat{s_N}\}$, we define the *average edit error* as the average distance between the generated shapes and the corresponding ground-truth shapes:

$$\text{A}_d = \frac{1}{N} \sum_{t=1}^{N} d(s_t, \widehat{s_t})$$

And the *final edit error* as the distance between the last shape in the chain and the corresponding ground-truth shape:

$$\text{F}_d = d(s_N, \widehat{s_N})$$

We report results with both metrics for $d = d_{\text{CD}}$ and $d = d_{\mathcal{L}_2}$.

Results. We provide qualitative results of edited pairs generated by our latent mapper in Figure 5. Overall, we observe that our method is able to generate plausible edits that are consistent with the input instruction, and preserve the identity of the input shape. However, our method is limited by the quality of the shape autoencoder. We notice that the PC-AE editor is unable to perform very fine-grained edits which are not properly captured by the autoencoder, while visible in the ground-truth shapes. We explore the possible causes of these limitations in Section 5.2.

We provide quantitative results for our chained editing metrics averaged across $|\mathcal{P}| \in \{10, 15, 20\}$ in Table 1, and detail per-chain length results in the appendix. Directly predicting the edited shape latent (DIRECTGEN) leads to poor edit predictions across all metrics, although increasing the number of layers may improve the results. Overall, decoupling the edit direction and magnitude leads to better results. Our coupled OURS$_{512\times4}$ variant performs better than the coupled LATEFUSION$_{512}$ variant on the Chamfer Distance, but worse on $\mathcal{L}_2$ distance.

Model	decoupled?	Averaged $\forall\lvert\mathcal{P}\rvert$			
		$\mathrm{F}_{\mathrm{CD}\times\mathbf{1e^4}}$	$\mathrm{A}_{\mathrm{CD}\times\mathbf{1e^4}}$	$\mathrm{F}_{\mathcal{L}_2}$	$\mathrm{A}_{\mathcal{L}_2}$
LateFusion$_{1024}$	✗	2.856	2.621	1.444	1.251
LateFusion$_{512}$	✗	2.719	2.609	1.462	1.290
LateFusion$_{256}$	✗	2.874	2.651	1.509	1.283
Ours$_{512\times8}$	✗	3.208	2.822	1.703	1.437
Ours$_{512\times4}$	✗	2.990	2.703	1.589	1.351
LateFusion$_{1024}$	✓	2.711	2.568	**1.405**	**1.245**
LateFusion$_{512}$	✓	3.002	2.708	1.451	1.254
LateFusion$_{256}$	✓	3.309	2.848	1.552	1.324
Ours$_{512\times8}$	✓	2.782	2.584	1.497	1.290
Ours$_{512\times4}$	✓	**2.670**	**2.524**	1.447	1.266

Table 1. **Chained shape editing ablation.** We report baseline results for our proposed chained shape editing task, averaged across all chain lengths $|\mathcal{P}| \in \{10, 15, 20\}$. Both the average final and average edit error are reported for the Chamfer Distance (CD) and $\mathcal{L}_2$ distance ($\mathcal{L}_2$) metrics.

Parameter	**Accuracy**
seat height	1.000
backrest curvature	1.000
object width/height/depth	0.978
seat roundness	0.978
top bar thickness/height	0.961
legs thickness	0.956
legs bending/curvature	0.944
adding/removing handle cushions	0.750
number of legs/backrest rails	0.663
legs roundness/indentation	0.587
Avg (Random)	0.884
Avg (Realistic)	0.969

Table 2. **Parameter-wise accuracy for shape edit recognition.** We report the accuracy of our edit detector for each parameter in the *random* set, alongside the average accuracy for the full *random* and *realistic* sets. Overall, we observe that our classifier is able to recognize edits corresponding to high change edits, but fails to differentiate between subtle changes.

5.2. Recognizing Shape Edits

Problem. To provide additional insight into the quality of our latent mapper and the difficulty of predicting specific edits, we train a binary classifier to discriminate from a pair of shapes (s_i, s_j) on whether the corresponding edit instruction $\mathbf{p_{ij}}$ was applied to s_i or s_j. We train a binary classifier $f_C : \mathcal{S} \times \mathcal{S} \mapsto \{0, 1\}$, where $f_C(s_i, s_j) = 1$ if the edit instruction $\mathbf{p_{ij}}$ was applied to s_i, and 0 otherwise.

Method. Similarly to our latent mapper architecture for PC-AE, we use an MLP-based shape latent encoder to encode the input shape latents into the bottleneck dimension, and project text features using a single linear layer to the same dimension. We then concatenate the two feature vectors and pass them through an MLP predictor outputting probabilities for the two classes s_i and s_j. We train our classifier using a binary cross-entropy loss.

Training and evaluation. We train our classifier on the *realistic* set and evaluate it on the full *random* shape set, which has a restricted vocabulary of edit instructions. We evaluate the accuracy of the classifier on the full set, as well as on subsets of the data corresponding to specific types of edits.

Results. In Table 2, we provide accuracy results for shape edit recognition, depending on the edit type. We also provide the global accuracy on the *random* and *realistic* sets. We observe that our classifier is able to recognize edits corresponding to shape parameters associated with a high degree of shape variation (like seat height and width/height of the shape), but struggles with parameters which are more subtle (like legs roundness, handle cushions). Our classifier is able to detect edits related to global width/height/depth adjustments with an accuracy of 97.8%. This accuracy drops to 94.4% for legs bending/curvature, which is a more subtle change in the shape, and to 58.7% for legs roundness/indentation. We also note that the model does not perform well on edits related to the number of parts, i.e. adding or removing a number k of legs or backrest rails.

We impute these discrepancies to several factors. The PC-AE shape autoencoder we employ may be unable to properly reconstruct fine-grained details in the ground-truth and edited shapes, which renders the task of detecting subtle changes more difficult. Pointcloud resolution is also a limitation, as fine-grained parameters like leg indentation may not be properly sampled. Finally, learning a latent mapper and a classifier that can accurately count the number of parts to add or remove may also be a difficult task of its own, and require more domain-specific inductive biases.

6. Conclusion

In conclusion, this paper introduces the ShapeWalk dataset, designed for advancing compositional shape editing guided by natural language instructions. The dataset comprises 158K unique shapes connected through 26K edit chains, synthesized from a realistic CAD-designed 3D dataset. Language instructions for applied edits are provided, alongside exact ground-truth edited shapes. Our method requires zero human annotation effort, and can be scaled up indefinitely. The usefulness of the dataset is demonstrated by training neural editor modules in the latent space of shape autoencoders [1]. Evaluation metrics inspired by trajectory prediction literature [4, 26] are introduced, offering a quantitative assessment of the quality of recursively edited shapes.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, 2018. 2, 6, 8

[2] Panos Achlioptas, Ian Huang, Minhyuk Sung, Sergey Tulyakov, and Leonidas Guibas. Shapetalk: A language dataset and framework for 3d shape edits and deformations. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2023. 6, 7

[3] Panos Achlioptas, Ian Huang, Minhyuk Sung, Sergey Tulyakov, and Leonidas Guibas. ShapeTalk: A language dataset and framework for 3d shape edits and deformations. In *CVPR*, 2023. 2, 3, 7

[4] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971, 2016. 2, 7, 8

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. In *arXiv*, 2015. 3, 6

[6] Prithiviraj Damodaran. Parrot: Paraphrase generation for nlu., 2021. 5

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. 6

[8] Rao Fu, Xiao Zhan, Yiwen Chen, Daniel Ritchie, and Srinath Sridhar. Shapecrafter: A recursive text-conditioned 3d shape generation model, 2023. 3, 6

[9] Aditya Ganeshan, R. Kenny Jones, and Daniel Ritchie. Improving unsupervised visual program inference with code rewriting families. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2023. 2

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 3, 4

[11] 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*, 39(6):1–20, 2020. 2

[12] 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*, 40(4), 2021. 2

[13] R. Kenny Jones, Homer Walke, and Daniel Ritchie. Plad: Learning to infer shape programs with pseudo-labels and approximate distributions. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022. 2

[14] R. Kenny Jones, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Transactions on Graphics*, 42(4), 2023. 2

[15] Yuchen Li, Ujjwal Upadhyay, Habib Slim, Ahmed Abdelreheem, Arpit Prajapati, Suhail Pothigara, Peter Wonka, and Mohamed Elhoseiny. 3D CoMPaT: Composition of Materials on Parts of 3D Things. In *ECCV*, 2022. 3, 4

[16] Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. Clip-mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 Conference Papers*. ACM, 2022. 2, 3

[17] Ofek Pearl, Itai Lang, Yuhua Hu, Raymond A. Yeh, and Rana Hanocka. Geocode: Interpretable shape programs. In *arXiv preprint arxiv:2212.11715*, 2022. 2, 4

[18] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. In *arXiv*, 2023. 5

[19] Daniel Ritchie, Paul Guerrero, R. Kenny Jones, Niloy J. Mitra, Adriana Schulz, Karl D. D. Willis, and Jiajun Wu. Neurosymbolic models for computer graphics. *Computer Graphics Forum*, 42(2), 2023. 2

[20] 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 *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022. 2

[21] Aditya Sanghi, Rao Fu, Vivian Liu, Karl D.D. Willis, Hooman Shayani, Amir H. Khasahmadi, Srinath Sridhar, and Daniel Ritchie. Clip-sculptor: Zero-shot generation of high-fidelity and diverse shapes from natural language. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2023. 2, 3

[22] Habib Slim, Xiang Li, Mahmoud Ahmed Yuchen Li, Mohamed Ayman, Ujjwal Upadhyay Ahmed Abdelreheem, Suhail Pothigara Arpit Prajapati, Peter Wonka, and Mohamed Elhoseiny. 3DCoMPaT++: An improved large-scale 3d vision dataset for compositional recognition. In *arXiv preprint arxiv:2212.11715*, 2023. 3, 4

[23] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *CoRR*, abs/1711.00937, 2017. 3

[24] Suchen Wang, Yueqi Duan, Henghui Ding, Yap-Peng Tan, Kim-Hui Yap, and Junsong Yuan. Learning transferable human-object interaction detector with natural language supervision. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022. 2

[25] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. In *SIGGRAPH*, 2019. 4

[26] Kaiping Xu, Zheng Qin, Guolong Wang, Kai Huang, Shuxiong Ye, and Huidi Zhang. *Collision-Free LSTM for Human Trajectory Prediction*, pages 106–116. 2018. 2, 7, 8

[27] Fenggen Yu, Zhiqin Chen, Manyi Li, Aditya Sanghi, Hooman Shayani, Ali Mahdavi-Amiri, and Hao Zhang. Capri-net: Learning compact cad shapes with adaptive primitive assembly. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022. 2

[28] Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics*, 42(4):1–16, 2023. 6

ShapeWalk: Compositional Shape Editing through Language-Guided Chains

Supplementary Material

Code and Dataset

Our dataset, generation code, model checkpoints and training scripts will be made available upon publication at:

https://shapewalk.github.io/

Appendix Overview

This supplementary document is organized as follows:

- Section 1 provides additional analysis of our latent editor models.
- Section 2 provides additional implementation details about our dataset generation and model training processes.
- Section 3 gives shape reconstruction examples.
- Section 4 showcases edit chain samples from our dataset, including edit chains for additional **table** and **vase** classes, and dataset statistics.
- Section 5 illustrates the sampling process for the 3D2VS latent diffusion model.

1. Analysis

We differentiate in the additional results in this section between $\mathrm{CD}-\textsc{Rec}$ and $\mathrm{CD}-\textsc{Real}$ metrics, which respectively measure the distance to the reconstructed edited shape and the real edited shape. The latter metric is more relevant for our task, as it measures the final editing ability of the model (reported in the main paper).

1.1. Chained Shape Editing

We provide detailed results for the chained shape editing task described in the main paper, in Table 2. We report baseline results for our proposed chained shape editing task, for chain lengths $|\mathcal{P}| \in \{10, 15, 20\}$. Overall, while no clear trend is observed for the Chamfer Distance, we observe that the average edit error for $\mathcal{L}_2$ distance decreases with longer chains. This could be explained by the fact that longer chains contain more sequences of fine-grained edits which are less likely to make the feature representations diverge abruptly. On the other hand, shorter chains are more likely to contain sequences of high-magnitude edits, which amplify the cumulative prediction error. In that sense, a parallel could be made with the long-horizon prediction problem in trajectory forecasting, where the prediction error accumulates over time [2].

1.2. Oracle Editing

We provide detailed results for the chained shape editing task when oracle magnitudes and directions are used, in Table 2. Overall, the biggest gain for PC-AE based models is observed when using oracle directions. This suggests that the main bottleneck for our models is the edit direction prediction task, which is more challenging than predicting the magnitude from the input prompt.

1.3. Predicting Edit Magnitudes

Our decoupled latent editor models predict both a normalized edit direction $\hat{v}_{ij}$ and an edit magnitude $\hat{m}_{ij}$ for each edit. We use this property to analyze the correlation between the predicted edit magnitudes and the ground truth edit intensities extracted from our metadata. We extract decoupled edit vector predictions from our models on the *random* subset, and compare them to the ground truth edit intensities. In order to do that, we min-max normalize the predicted edit magnitudes and bin them in the $[1, 9]$ interval. Ground-truth edit intensities are extracted from the ground-truth edit vectors, similarly by discretizing the magnitude of changes in the same interval. We plot confusion matrices comparing these predicted discretized edit magnitudes to the ground truth edit intensities in Figure 1. We show results for three decoupled baselines: $\textsc{LateFusion}_{1024}$, $\textsc{LateFusion}_{512}$, and $\textsc{Ours}_{512\times4}$.

Overall, we observe a strong correlation between the predicted edit magnitudes and the ground truth edit intensities across all models. However all models tend to deviate from the ideal diagonal magnitude predictions. We remark also that the $\textsc{Ours}_{512\times4}$ model tends to deviate from the diagonal and avoids predicting the maximum possible magnitude range. One way to alleviate these biases could be to introduce explicit supervision to the edit magnitude prediction task using the ground truth edit magnitude labels. For example, a contrastive loss could incentivize the model to predict edit magnitudes that align with the ground truth ranking of edit intensities.

2. Implementation Details

2.1. Shape Chain Generation

We detail in Figure 9 the process of sampling parameter chains from the parameter tree. Each parameter vector θ_i can be represented as a dependency tree. We start by sampling a parameter tree θ_0 using regressed parameters from a real **source** shape in the *realistic* subset. We then successively sample parameters to edit by interpolating towards a **target** shape in the *realistic* setting[1].

[1] In the *random* setting, the source shape and subsequent parameters are sampled randomly.

Model	decoupled?	$\lvert\mathcal{P}\rvert$ = 10						$\lvert\mathcal{P}\rvert$ = 15						$\lvert\mathcal{P}\rvert$ = 20					
		$\mathrm{F}_{\mathrm{CD-Rec}}$	$\mathrm{A}_{\mathrm{CD-Rec}}$	$\mathrm{F}_{\mathrm{CD-Real}}$	$\mathrm{A}_{\mathrm{CD-Real}}$	$\mathrm{F}_{\mathcal{L}_2}$	$\mathrm{A}_{\mathcal{L}_2}$	$\mathrm{F}_{\mathrm{CD-Rec}}$	$\mathrm{A}_{\mathrm{CD-Rec}}$	$\mathrm{F}_{\mathrm{CD-Real}}$	$\mathrm{A}_{\mathrm{CD-Real}}$	$\mathrm{F}_{\mathcal{L}_2}$	$\mathrm{A}_{\mathcal{L}_2}$	$\mathrm{F}_{\mathrm{CD-Rec}}$	$\mathrm{A}_{\mathrm{CD-Rec}}$	$\mathrm{F}_{\mathrm{CD-Real}}$	$\mathrm{A}_{\mathrm{CD-Real}}$	$\mathrm{F}_{\mathcal{L}_2}$	$\mathrm{A}_{\mathcal{L}_2}$
LateFusion$_{1024}$	✗	1.751	1.431	3.006	2.756	1.913	1.707	2.150	1.579	2.941	2.925	1.366	1.241	1.591	0.944	2.620	2.182	1.054	**0.805**
LateFusion$_{512}$	✗	1.802	1.413	3.031	2.801	2.018	1.770	1.804	1.469	2.824	2.755	1.309	1.221	1.414	1.005	2.301	2.272	1.059	0.880
LateFusion$_{256}$	✗	1.807	1.486	3.026	2.928	2.079	1.810	2.034	1.380	2.852	2.718	1.342	1.203	1.737	1.018	2.743	2.306	1.106	0.838
Ours$_{512\times 8}$	✗	2.240	1.665	3.680	3.169	2.311	2.022	2.471	1.664	3.347	2.956	1.591	1.351	1.571	0.992	2.599	2.341	1.207	0.937
Ours$_{512\times 4}$	✗	1.751	1.494	3.236	2.859	2.168	1.884	2.378	1.540	3.098	2.850	1.432	1.271	1.659	1.087	2.636	2.401	1.166	0.898
LateFusion$_{1024}$	✓	1.773	1.452	3.066	2.794	2.002	1.734	1.704	1.290	2.787	2.667	**1.212**	**1.184**	1.502	1.049	2.279	2.242	**0.999**	0.817
LateFusion$_{512}$	✓	1.687	1.430	**2.694**	**2.698**	**1.883**	**1.672**	1.978	1.328	3.089	2.749	1.300	1.190	2.149	1.307	3.224	2.676	1.170	0.900
LateFusion$_{256}$	✓	2.057	1.555	3.071	2.824	2.043	1.743	2.306	1.630	3.342	2.987	1.443	1.305	2.397	1.394	3.514	2.734	1.170	0.923
Ours$_{512\times 8}$	✓	1.827	1.358	3.243	2.858	2.097	1.809	1.982	1.331	2.918	2.679	1.359	1.234	**1.331**	**0.865**	**2.187**	2.214	1.035	0.826
Ours$_{512\times 4}$	✓	**1.647**	**1.348**	3.028	2.826	1.961	1.765	**1.698**	**1.240**	**2.713**	**2.582**	1.328	1.211	1.450	0.884	2.268	**2.163**	1.053	0.821

Table 1. **Chained shape editing ablation.** We report detailed baseline results for our proposed chained shape editing task, for chain lengths $|\mathcal{P}| \in \{10, 15, 20\}$, using the PC-AE trained latent editors. Both the average final error and average edit error are reported for the Chamfer Distance (CD) and $\mathcal{L}_2$ distance ($\mathcal{L}_2$) metrics. We differentiate between distances to the reconstructed edited shape ($\mathrm{CD} - \mathrm{Rec}$) and distances to the real edited shape ($\mathrm{CD} - \mathrm{Real}$), which is the most relevant metric for our task. We highlight in grey the model we select for our qualitative results.

Model	decoupled?	$\lvert\mathcal{P}\rvert$ = 10						$\lvert\mathcal{P}\rvert$ = 15						$\lvert\mathcal{P}\rvert$ = 20					
		$\mathrm{F}_{\mathrm{CD-Rec}}$	$\mathrm{A}_{\mathrm{CD-Rec}}$	$\mathrm{F}_{\mathrm{CD-Real}}$	$\mathrm{A}_{\mathrm{CD-Real}}$	$\mathrm{F}_{\mathcal{L}_2}$	$\mathrm{A}_{\mathcal{L}_2}$	$\mathrm{F}_{\mathrm{CD-Rec}}$	$\mathrm{A}_{\mathrm{CD-Rec}}$	$\mathrm{F}_{\mathrm{CD-Real}}$	$\mathrm{A}_{\mathrm{CD-Real}}$	$\mathrm{F}_{\mathcal{L}_2}$	$\mathrm{A}_{\mathcal{L}_2}$	$\mathrm{F}_{\mathrm{CD-Rec}}$	$\mathrm{A}_{\mathrm{CD-Rec}}$	$\mathrm{F}_{\mathrm{CD-Real}}$	$\mathrm{A}_{\mathrm{CD-Real}}$	$\mathrm{F}_{\mathcal{L}_2}$	$\mathrm{A}_{\mathcal{L}_2}$
LateFusion$_{1024}$	✓	1.773	1.452	3.066	2.794	2.002	1.734	1.704	1.290	2.787	2.667	1.212	1.184	1.502	1.049	2.279	2.242	0.999	0.817
$+\ \mathbb{O}$ Magnitude	✓	1.118	0.941	2.323	2.242	1.592	1.437	1.037	0.910	2.106	2.142	0.978	0.996	0.608	0.631	1.442	1.892	0.784	0.714
$+\ \mathbb{O}$ Direction	✓	0.688	0.360	2.109	2.054	0.822	0.551	0.360	0.279	1.903	1.944	0.410	0.329	0.137	0.341	1.358	1.747	0.207	0.316
LateFusion$_{512}$	✓	1.687	1.430	2.694	2.698	1.883	1.672	1.978	1.328	3.089	2.749	1.300	1.190	2.149	1.307	3.224	2.676	1.170	0.900
$+\ \mathbb{O}$ Magnitude	✓	1.206	1.017	2.328	2.300	1.628	1.445	1.192	1.015	2.241	2.311	1.027	1.027	0.777	0.762	1.727	2.039	0.816	0.731
$+\ \mathbb{O}$ Direction	✓	0.699	0.386	2.136	2.082	0.805	0.561	0.507	0.356	2.134	2.020	0.500	0.384	0.094	0.345	1.347	1.776	0.172	0.319
LateFusion$_{256}$	✓	2.057	1.555	3.071	2.824	2.043	1.743	2.306	1.630	3.342	2.987	1.443	1.305	2.397	1.394	3.514	2.734	1.170	0.923
$+\ \mathbb{O}$ Magnitude	✓	1.150	0.952	2.337	2.280	1.636	1.457	1.272	1.104	2.477	2.429	1.159	1.085	0.684	0.734	1.583	2.072	0.770	0.736
$+\ \mathbb{O}$ Direction	✓	0.719	0.423	2.185	2.108	0.846	0.610	0.747	0.449	2.267	2.112	0.695	0.468	0.267	0.463	1.422	1.804	0.309	0.402

Table 2. **Oracle editing results.** We report detailed baseline results when oracle magnitudes and directions are used for the chained shape editing task, on all LateFusion models. We differentiate between distances to the reconstructed edited shape ($\mathrm{CD} - \mathrm{Rec}$) and distances to the real edited shape ($\mathrm{CD} - \mathrm{Real}$), which is the most relevant metric for our task. Overall, the biggest gain for PC-AE based models is observed when using oracle directions.

When boolean parameters with children are triggered, we sample a random value for each child parameter. For scalar parameters with a continuous domain, we sample a random value from a discretized version of the parameter's range. After generating each parameter vector θ_i, we pass it through a geometry checker (implemented in [12]) ensuring that the resulting shape is valid. If the shape is invalid, the whole chain is discarded and we start over from a different starting pair. Otherwise, the corresponding mesh is synthesized using the shape program ϕ_Θ and added to the chain.

2.2. Text Instructions Generation

We detail in Figure 2 the process of synthesizing text instructions for the chained shape editing task. Starting from an edit vector, we map the edit intensity and the parameter name to a predefined vocabulary set which is randomly sampled. Optionally, the generated instruction is paraphrased using a pre-trained language model. Our instruction generation method is completely automatic and does not require any human annotation or additional data.

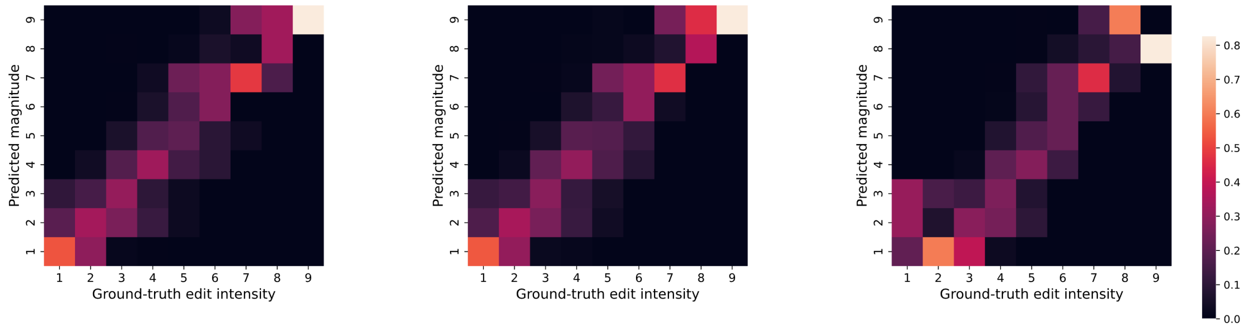
2.3. Architecture details

Rendered View Feature Extractor. We use a ResNet-50 [7] model pre-trained on ImageNet [5] to extract rendered view features from the input shapes. Since all synthetic meshes are centered at the origin and aligned, we do not require multi-view features to ensure invariance to rotation. Every mesh is rotated around the z-axis by an angle $\theta = \frac{\pi}{8}$, assigned a base RGB color and rendered from a static viewpoint using the trimesh [4] library.

Autoencoders. We encode pointclouds using a PC-AE [1] model pre-trained on the ShapeNet [3] dataset with a latent dimension $d = 256$. We also use a 3D2VS [17] model also pre-trained on ShapeNet with a latent dimension $d = 512 \times 8$. Note that our method is agnostic to the choice of autoencoder model, and can be adapted to any other shape or pointcloud representation.

PC-AE Latent Editors. We detail here the architecture of our latent editor models.

- **LateFusion$_X$** is composed of a shape latent encoder,



a) LateFusion$_{1024}$ b) LateFusion$_{512}$ c) Ours$_{512\times 4}$

Figure 1. **Comparing predicted edit magnitude to ground truth edit intensity.** We plot confusion matrices comparing the predicted edit magnitudes binned in the interval $[1, 9]$ to the ground truth edit intensities extracted from our metadata. We show results for three decoupled baselines: LateFusion$_{1024}$, LateFusion$_{512}$, and Ours$_{512\times 4}$. Overall, we observe a strong correlation between the predicted edit magnitudes and the ground truth edit intensities. However, a bias can be observed in the Ours$_{512\times 4}$ model, which tends to deviate from the diagonal predictions, and avoids predicting the maximum possible magnitude range.

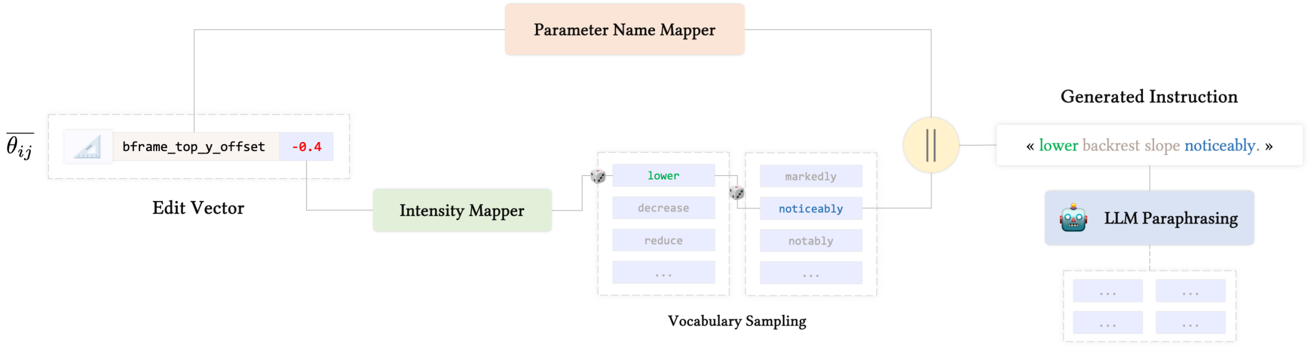


Figure 2. **Synthesizing edit instructions.** Our method generates synthetic text instructions by first applying rule-based generation to translate parameter changes into natural language. The edit intensity is mapped to a natural language intensity depending on the parameter type. To add diversity, the vocabulary describing the edit is randomly sampled, edit directions are randomly inverted, and a paraphrasing transformer model is optionally employed to augment the final instructions.

an edit direction prediction module, and an edit magnitude prediction module. The *shape latent encoder* is a 2-layer MLP with hidden dimensions $[256, 256]$. We use ReLU activations for all layers, and a linear activation for the output layer. The *edit direction prediction module* is a 4-layer MLP with hidden dimensions $[X, 256, 256, 256]$. Since we normalize the predicted edit vector, we remove the final bias term from the output layer. We use batch normalization [8] for all layers except the output layer. The *edit magnitude prediction module* is a 3-layer MLP with hidden dimensions $[256, 128, 64]$ respectively.
For all modules, we use dropout [14] with a probability of 0.1 for all layers except the output layer.

- **Ours**$_{512\times X}$ only uses the edit direction prediction module which is an X-layer MLP with hidden dimensions of the size of the latent space of the autoencoder. We use batch normalization [8] for all layers except the output layer, dropout with a probability of 0.2, and ReLU activations for all layers. We separately predict the edit magnitude and edit direction using the same modules as LateFusion.

We illustrate in Figure 4 the architecture of the latent editor diffusion model.

Text Encoder Model. We use a pre-trained BERT [6] model to encode the language instructions into a 768-dimensional feature vector. We employ the base uncased model with 12 layers, 12 self-attention heads, and $110M$ parameters. Training with larger instances of BERT or with LLM-extracted text features could potentially improve the generalization ability of our models on unseen language instructions.

3D2VS Latent Editors. We fine-tune a latent diffusion

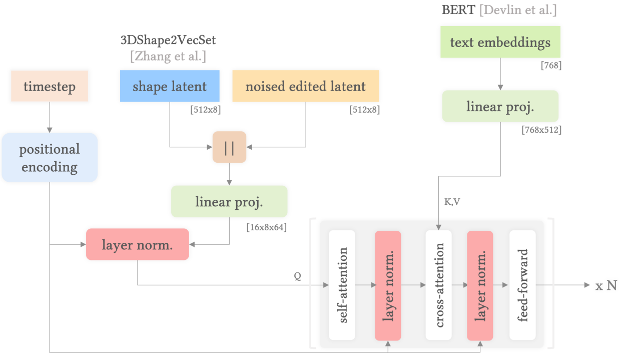


Figure 3. **Diffusion-based latent editor.** We illustrate the architecture of the transformer-based latent editor diffusion model used to edit 3D2VS [17] latents. || indicates concatenation of the input features.

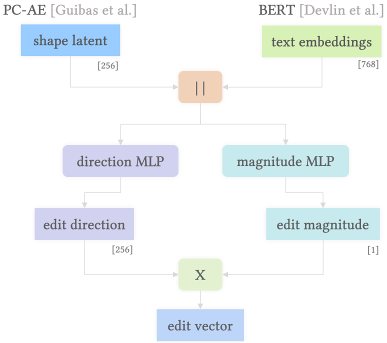


Figure 4. **MLP-based latent editor.** We illustrate the architecture of the MLP-based latent editor used to edit PC-AE [1] latents, for the decoupled magnitude and direction prediction models. || indicates concatenation of the input features.

model trained in the space of a frozen 3D2VS [17] auto-encoder. The best performing model uses a depth of 24 layers with 8 channels and a latent dimension of 512 × 8, and conditions the model on text features by feeding them as keys and values to the cross-attention layers. We illustrate in Figure 3 the architecture of the latent editor diffusion model.

2.4. Training

Shape Sampling. We sample $N = 4096$ points from each synthesized mesh using the triangle point sampling scheme [16]. Pointclouds sampled from synthetic shapes are centered at the origin, normalized to the unit sphere, and rescaled alongside each axis to align with ShapeNet statistics. A minor downside of normalizing shapes is that the model will only be able to learn to apply edits relative to the scale of the input shapes.

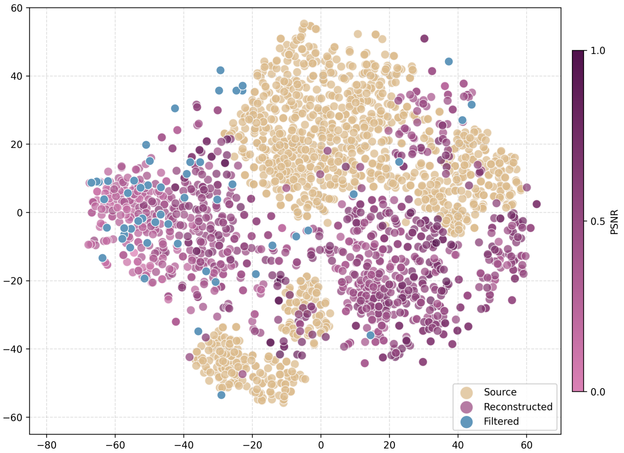


Figure 5. **t-SNE embeddings of shape reconstructions.** We plot t-SNE embeddings [15] of the rendered view features of the input shapes and their reconstructions in the space of the rendered view feature extractor ϕ_R. We also provide the PSNR ratio between input and reconstructed shape rendered views.

PC-AE Latent Editor. We train our models using the Adam optimizer [10] with a learning rate of 1×10^{-4}, and an effective bath size of 128. The learning rate is linearly increased from 1×10^{-6} to 1×10^{-4} during the first 8 epochs, and then gradually decayed back to 1×10^{-6} following a cosine annealing schedule [11]. We train our models on two NVIDIA V100 GPUs with 32GB of memory each. Training on the full *realistic* set takes around 60 minutes, with pre-extracted text and pointcloud features. Almost all models converge within 50 epochs, and we use the best model checkpoint based on the validation loss for all experiments.

3D2VS Latent Editor. To fine-tune the latent diffusion models, we use gradient clipping and a half-cycle cosine after a warmup of 20 epochs. We train the models for 200 epochs with a batch size of 32 on eight NVIDIA V100 GPUs with 32GB of memory each. Training and sampling strategies are the same as in [9, 17].
All text and pointcloud features are pre-extracted and cached to disk to speed up training.

3. Shape Reconstructions

We illustrate in Figure 12 the top-8 and bottom-8 shape reconstructions from the 3DCoMPaT [13] dataset into the GeoCode representation, ranked by rendered view feature similarity. Overall, we observe that the reconstructions are of high quality, and that the model is able to capture the main shape features of the input shapes. The worst reconstructions are discarded from the shape interpolation process as they do not generally lead to realistic reconstructed shapes.

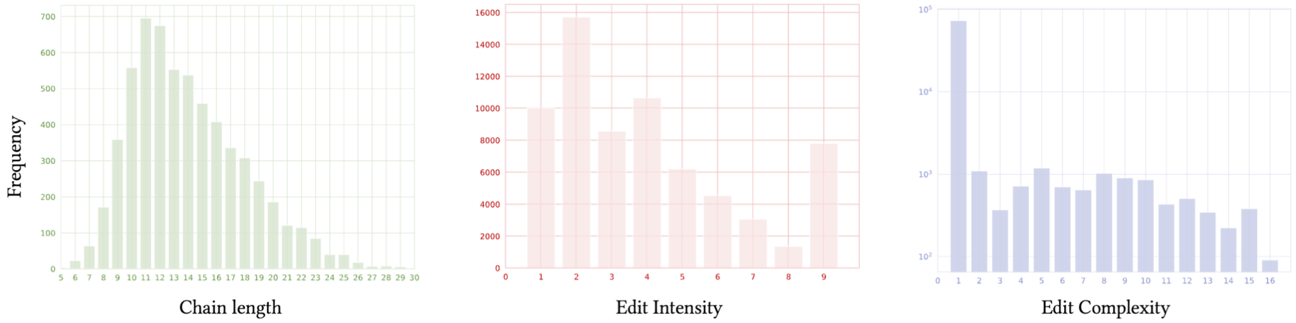


Figure 6. **ShapeWalk dataset statistics.** We plot the distribution of the number of shapes per chain (*left*), of the intensity of edits (*middle*), and of the complexity of edits (*right*). For the edit complexity, note that the frequency is provided in log-scale. The vast majority of edits in the *realistic* subset are granular, affecting a single shape parameter.

In order to further investigate the quality of the reconstructions, we compute t-SNE embeddings [15] of the rendered view features of the input shapes and their corresponding reconstructions in the space of ϕ_R, the rendered view feature extractor. We show the results in Figure 5. While reconstructed (in purple) and source shapes (in yellow) from the 3DCoMPaT [13] dataset form two distinct clusters, we observe that the reconstructions are generally close to the input shapes in the feature space. The filtered reconstructions (in blue) are more likely to be outliers in the feature space, and are thus discarded from the shape interpolation process to avoid unrealistic shape interpolations.

4. Dataset Insights

4.1. Shape Chains

In Figures 10, and 11, we showcase sampled shape chains from our dataset truncated to the first $N = 9$ shapes. For each chain edge, we show the corresponding language instruction describing the parameter changes necessary to transition from one shape to the next.

4.2. Dataset Statistics

In Figure 6, we plot the distribution of the number of shapes per chain (*left*), of the intensity of edits (*middle*), and of the complexity of edits (*right*) in the *realistic* subset[2]. The edit complexity corresponds to the number of shape parameters affected by the edit. For edit complexity, the frequency is provided in log-scale. We observe that the vast majority of edits in the *realistic* subset are granular, and affect a single shape parameter. Chain lengths range from $N = 6$ to $N = 29$ shapes, with an expected length around $N = 15$ shapes. The intensity of edits is not uniformly distributed and skews heavily towards low-intensity edits, which is expected.

5. Diffusion-based Editor Generation Process

We illustrate in Figure 7 the process of generating a shape chain using a diffusion-based latent editor model. We showcase the generation of a shape chain from a source shape to a target shape using our latent editor operating in the space of the 3D2VS autoencoder, illustrated in Figure 3. We show 32 intermediate shapes generated by the diffusion model for each sampling step.

[2] Note that we do not show statistics for the *random* subset as all parameters including chain length and edit intensity are static and determined at generation.

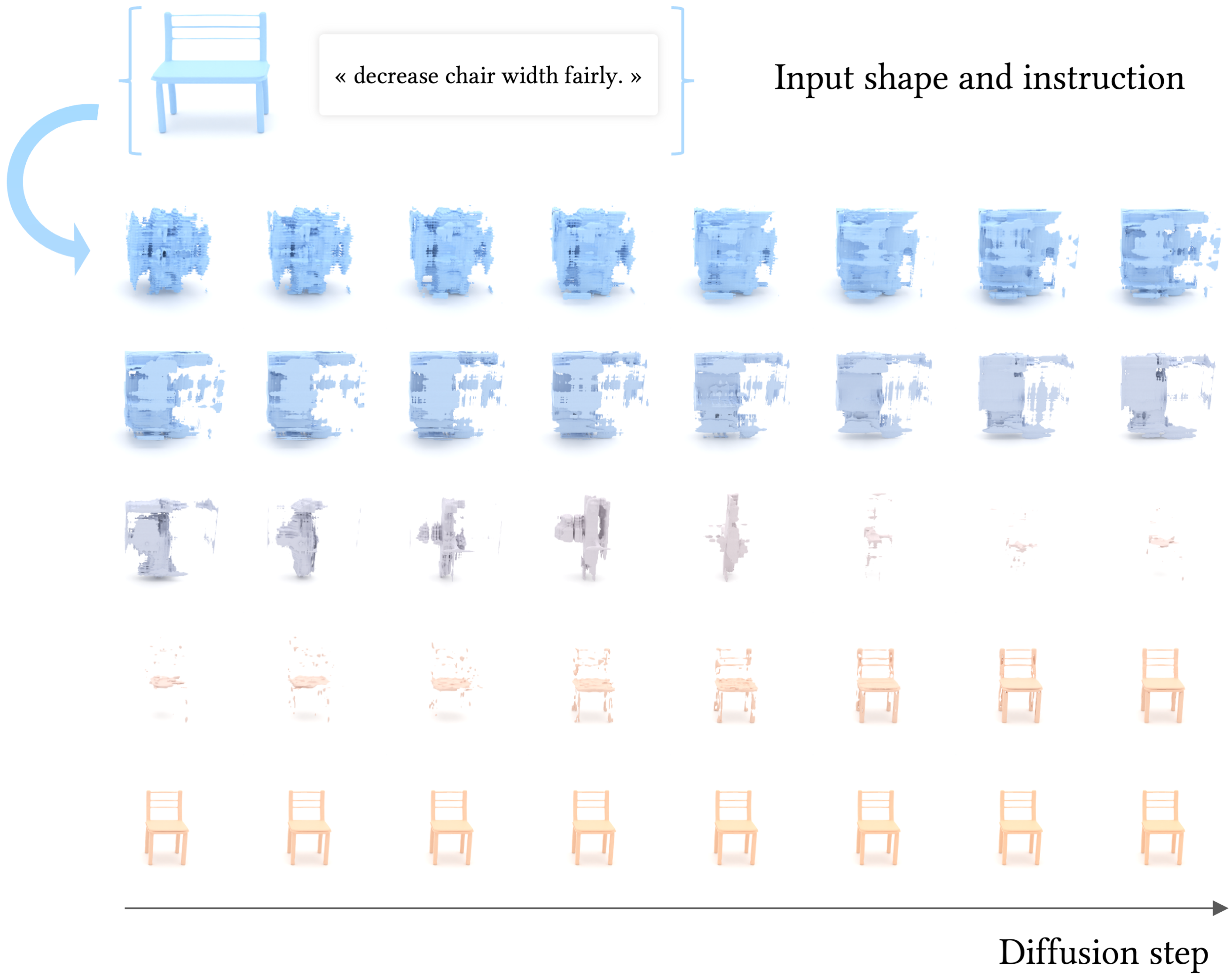


Figure 7. **Diffusion-based shape chain generation.** We illustrate in this figure the process of generating a shape using a diffusion-based latent editor model, for a source shape (**blue**) and a text instruction, to obtain a final target shape (**orange**) after $T = 32$ diffusion steps.

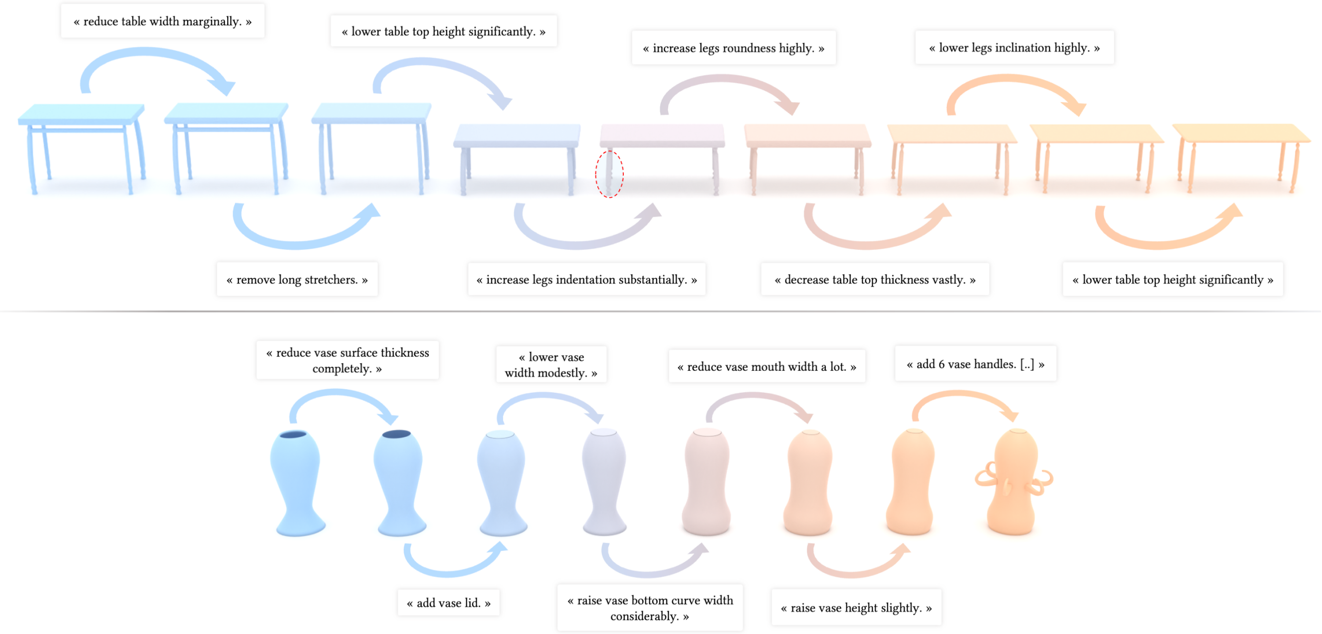

« reduce table width marginally. »
« remove long stretchers. »
« lower table top height significantly. »
« increase legs indentation substantially. »
« increase legs roundness highly. »
« decrease table top thickness vastly. »
« lower legs inclination highly. »
« lower table top height significantly »
« reduce vase surface thickness completely. »
« add vase lid. »
« lower vase width modestly. »
« raise vase bottom curve width considerably. »
« reduce vase mouth width a lot. »
« raise vase height slightly. »
« add 6 vase handles. [..] »

Figure 8. **Chain samples for additional classes.** We showcase in this figure additional chain samples for the *realistic* subset, for the *table* and *vase* classes. Our method is able to generate realistic shape chains for a variety of classes, given support for the corresponding shape programs. In our case, GeoCode [12] supports the classes we use in this work. Extending our method to new classes will require extending these underlying shape programs.

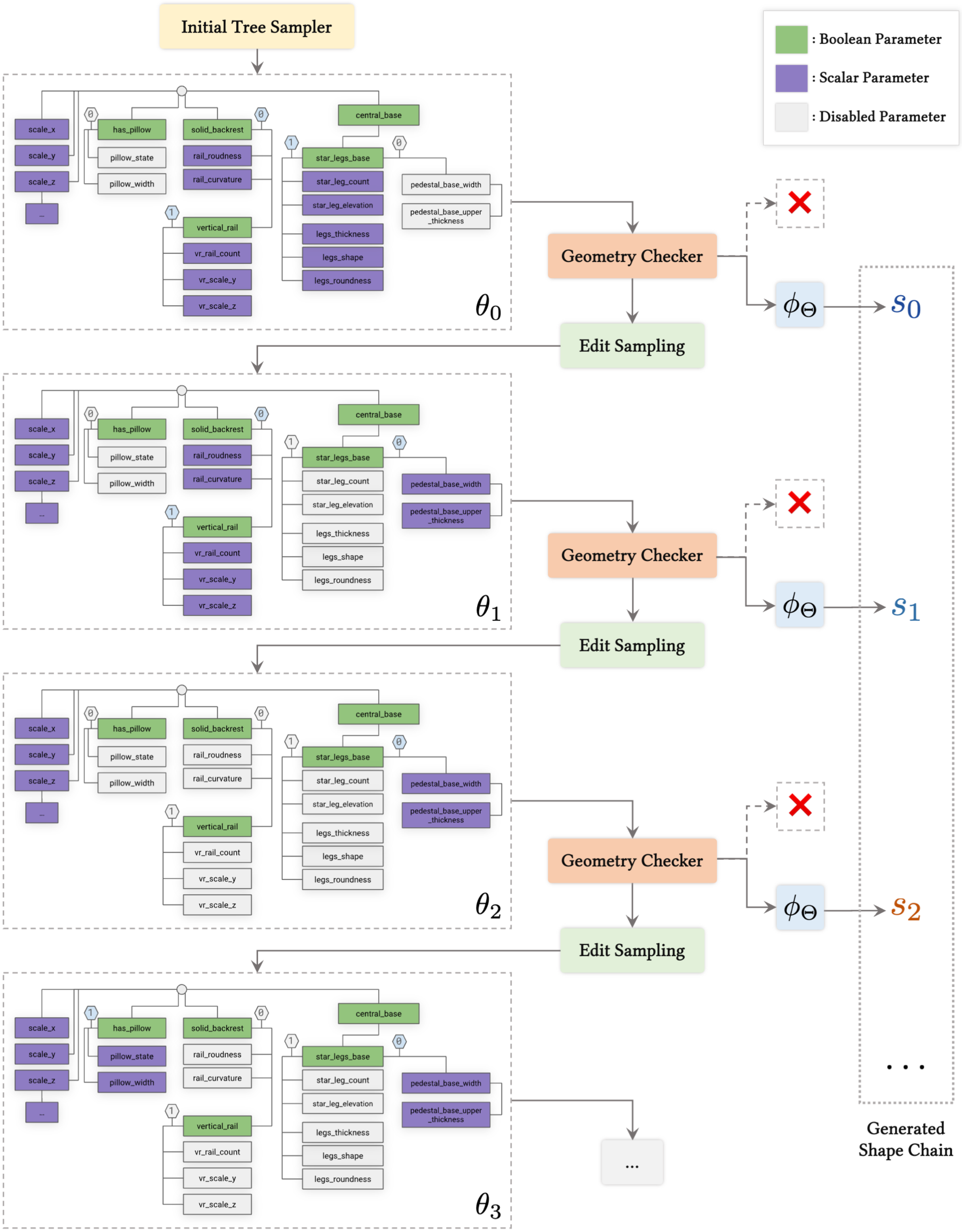


Figure 9. **Parameter tree sampling.** We detail in this figure the process of sampling parameter chains from the parameter tree. Each parameter vector θ_i can be represented as a dependency tree. We start by sampling a random parameter tree θ_1 from the root node of the tree (or using regressed parameters from a real shape in the *realistic* subset). Edit parameters are then sampled randomly (or using interpolation for the *realistic* set) from the tree until the desired chain length is reached (or until the target shape is reached).

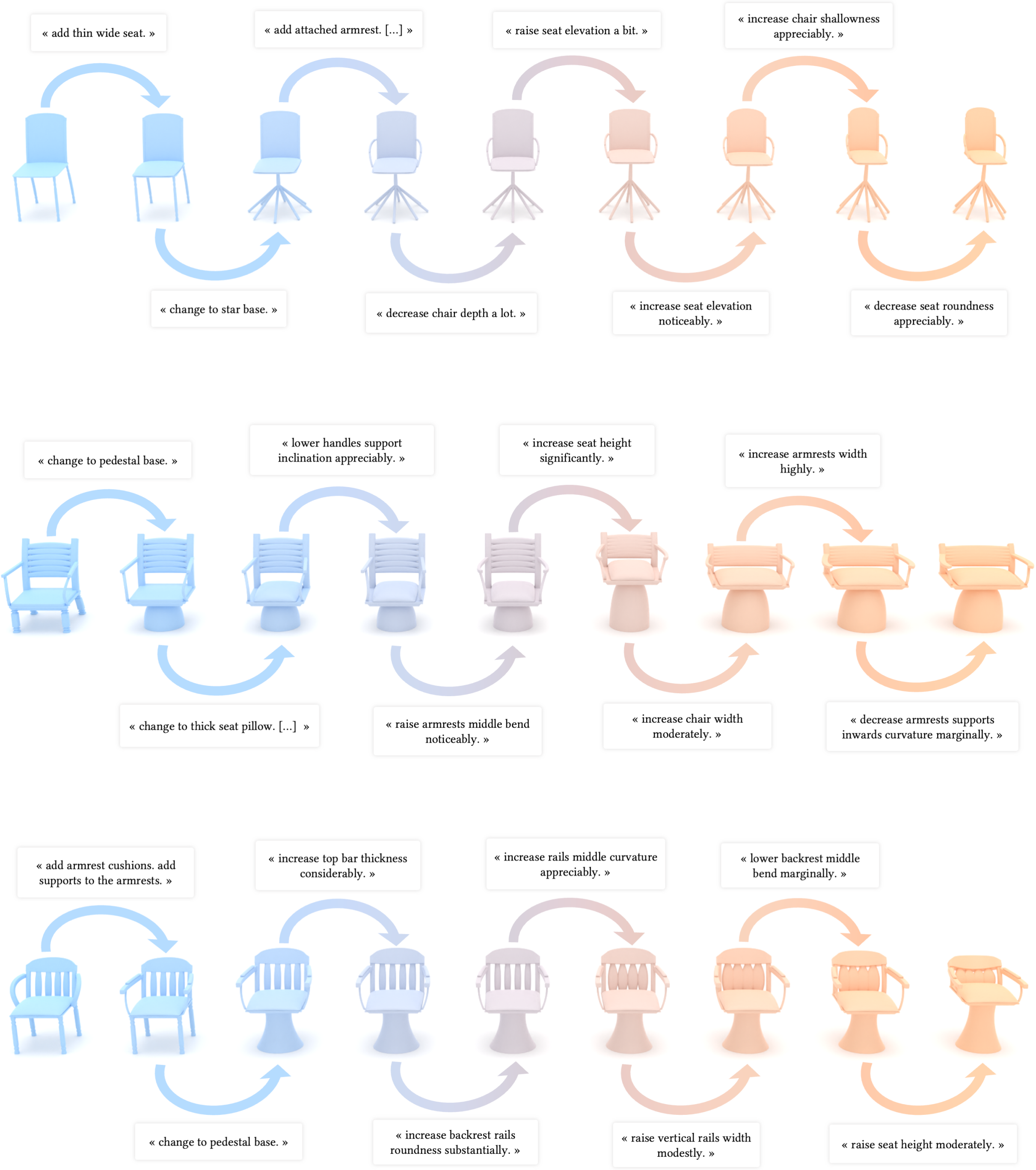

« add thin wide seat. »
« change to star base. »
« add attached armrest. [...] »
« decrease chair depth a lot. »
« raise seat elevation a bit. »
« increase seat elevation noticeably. »
« increase chair shallowness appreciably. »
« decrease seat roundness appreciably. »
« change to pedestal base. »
« change to thick seat pillow. [...] »
« lower handles support inclination appreciably. »
« raise armrests middle bend noticeably. »
« increase seat height significantly. »
« increase chair width moderately. »
« increase armrests width highly. »
« decrease armrests supports inwards curvature marginally. »
« add armrest cushions. add supports to the armrests. »
« change to pedestal base. »
« increase top bar thickness considerably. »
« increase backrest rails roundness substantially. »
« increase rails middle curvature appreciably. »
« raise vertical rails width modestly. »
« lower backrest middle bend marginally. »
« raise seat height moderately. »

Figure 10. **Additional dataset samples.** We showcase sampled chains from our dataset truncated to the first $N = 9$ shapes. We color shapes based on their proximity to the starting shape (**blue**), and the ending shape (**orange**). For each chain edge, we show the corresponding language instruction describing the parameter changes necessary to transition from one shape to the next. All shapes are normalized to the unit cube and centered at the origin.

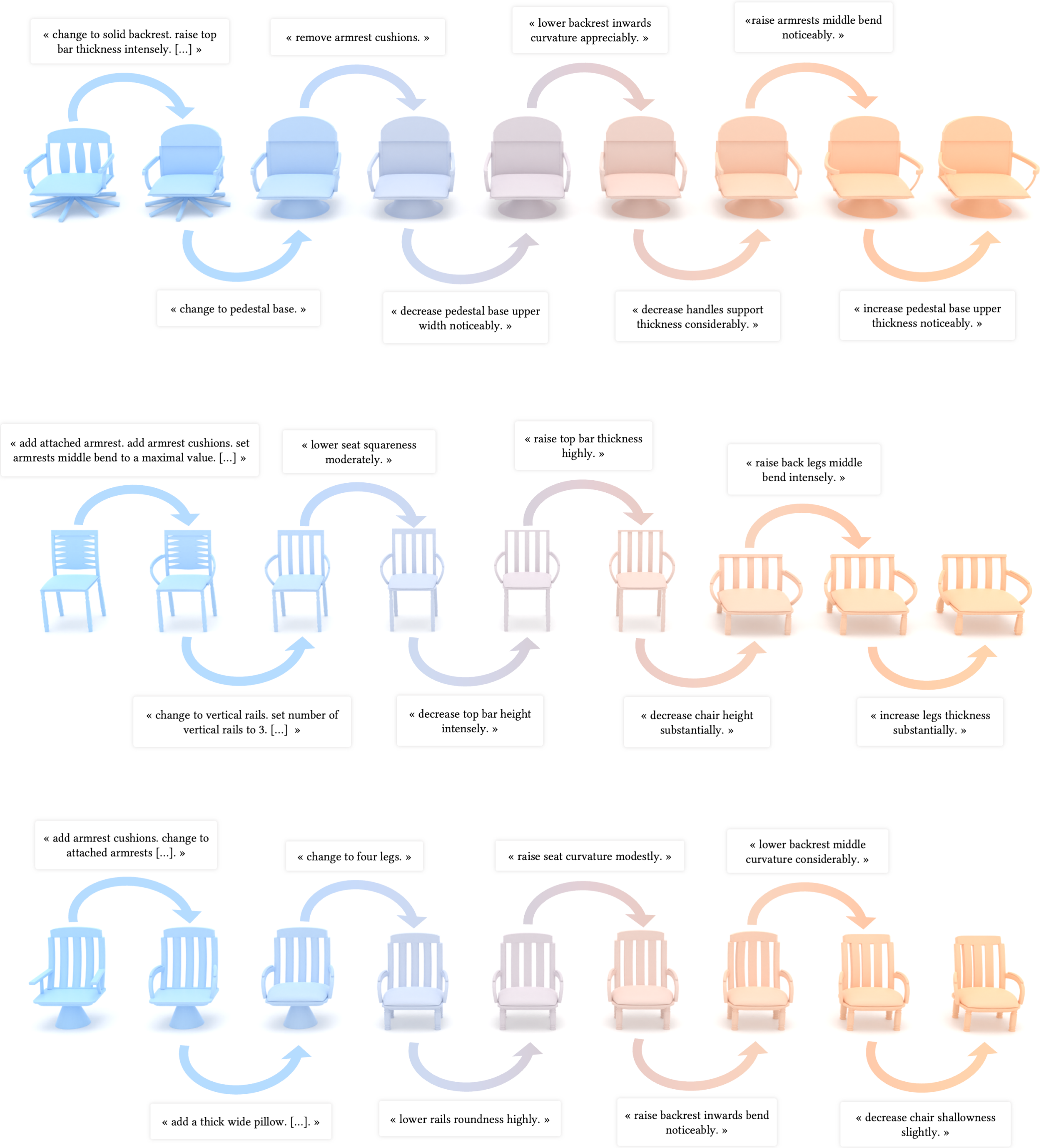

« change to solid backrest. raise top bar thickness intensely. [...] »
« change to pedestal base. »
« remove armrest cushions. »
« decrease pedestal base upper width noticeably. »
« lower backrest inwards curvature appreciably. »
« decrease handles support thickness considerably. »
«raise armrests middle bend noticeably. »
« increase pedestal base upper thickness noticeably. »
« add attached armrest. add armrest cushions. set armrests middle bend to a maximal value. [...] »
« change to vertical rails. set number of vertical rails to 3. [...] »
« lower seat squareness moderately. »
« decrease top bar height intensely. »
« raise top bar thickness highly. »
« decrease chair height substantially. »
« raise back legs middle bend intensely. »
« increase legs thickness substantially. »
« add armrest cushions. change to attached armrests [...]. »
« add a thick wide pillow. [...]. »
« change to four legs. »
« lower rails roundness highly. »
« raise seat curvature modestly. »
« raise backrest inwards bend noticeably. »
« lower backrest middle curvature considerably. »
« decrease chair shallowness slightly. »

Figure 11. **Additional dataset samples.** We showcase sampled chains from our dataset truncated to the first $N = 9$ shapes. We color shapes based on their proximity to the starting shape (**blue**), and the ending shape (**orange**). For each chain edge, we show the corresponding language instruction describing the parameter changes necessary to transition from one shape to the next. All shapes are normalized to the unit cube and centered at the origin.

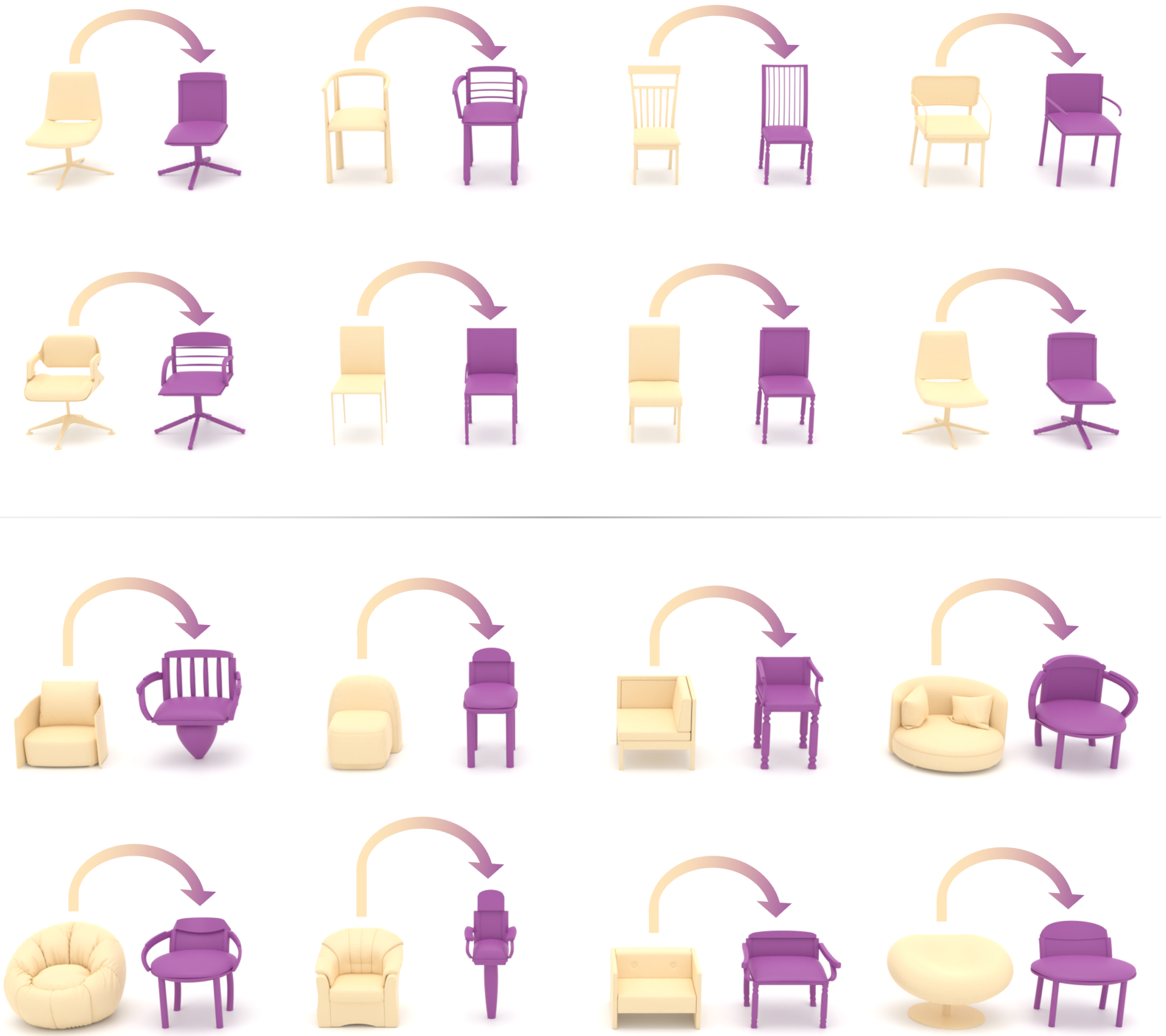


Figure 12. **Shape reconstruction examples.** We illustrate in this figure the original shapes from the 3DCoMPaT [13] dataset (**yellow**), and their corresponding reconstructions into the GeoCode representation (**purple**). In the top two rows, we show the top-8 reconstructions ranked by rendered view feature similarity, and the bottom-8 reconstructions ranked by rendered view feature similarity in the bottom two rows. While top reconstructions are generally of high quality, the bottom reconstructions are discarded from the shape interpolation process as they do not generally lead to realistic reconstructed shapes. We discard these shapes to avoid unrealistic shape interpolations in our shape chain generation process.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, 2018. 2, 4

[2] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971, 2016. 1

[3] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. In *arXiv*, 2015. 2

[4] Mike Dawson-Haggerty. Trimesh [computer software]. https://github.com/mikedh/trimesh, 2019. 2

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 2

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. 3

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 2

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 3

[9] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Proc. NeurIPS*, 2022. 4

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015. 4

[11] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. 4

[12] Ofek Pearl, Itai Lang, Yuhua Hu, Raymond A. Yeh, and Rana Hanocka. Geocode: Interpretable shape programs. In *arXiv preprint arxiv:2212.11715*, 2022. 2, 7

[13] Habib Slim, Xiang Li, Mahmoud Ahmed Yuchen Li, Mohamed Ayman, Ujjwal Upadhyay Ahmed Abdelreheem, Suhail Pothigara Arpit Prajapati, Peter Wonka, and Mohamed Elhoseiny. 3DCoMPaT++: An improved large-scale 3d vision dataset for compositional recognition. In *arXiv preprint arxiv:2212.11715*, 2023. 4, 5, 11

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. 3

[15] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. 4, 5

[16] Eric W. Weisstein. Triangle point picking. MathWorld–A Wolfram Web Resource. 4

[17] Biao Zhang, Jiapeng Tang, Matthias Nießner, and Peter Wonka. 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Transactions on Graphics*, 42(4):1–16, 2023. 2, 4