DOI: 10.1111/cgf.15276

COMPUTER GRAPHICS *forum*
Volume 44 (2025), number 1, e15276

GeoCode: Interpretable Shape Programs

Ofek Pearl,[1,2] Itai Lang,[2] Yuhua Hu,[2] Raymond A. Yeh[3] and Rana Hanocka[2]

[1]School of Electrical Engineering, Tel Aviv University, Tel Aviv-Yafo, Israel
ofekpearl@mail.tau.ac.il
[2]Department of Computer Science, University of Chicago, Chicago, USA
{itailang, katehu, ranahanocka}@uchicago.edu
[3]Department of Computer Science, Purdue University, West Lafayette, USA
rayyeh@purdue.edu

Abstract

The task of crafting procedural programs capable of generating structurally valid 3D shapes easily and intuitively remains an elusive goal in computer vision and graphics. Within the graphics community, generating procedural 3D models has shifted to using node graph systems. They allow the artist to create complex shapes and animations through visual programming. Being a high-level design tool, they made procedural 3D modelling more accessible. However, crafting those node graphs demands expertise and training. We present GeoCode, a novel framework designed to extend an existing node graph system and significantly lower the bar for the creation of new procedural 3D shape programs. Our approach meticulously balances expressiveness and generalization for part-based shapes. We propose a curated set of new geometric building blocks that are expressive and reusable across domains. We showcase three innovative and expressive programs developed through our technique and geometric building blocks. Our programs enforce intricate rules, empowering users to execute intuitive high-level parameter edits that seamlessly propagate throughout the entire shape at a lower level while maintaining its validity. To evaluate the user-friendliness of our geometric building blocks among non-experts, we conduct a user study that demonstrates their ease of use and highlights their applicability across diverse domains. Empirical evidence shows the superior accuracy of GeoCode in inferring and recovering 3D shapes compared to an existing competitor. Furthermore, our method demonstrates superior expressiveness compared to alternatives that utilize coarse primitives. Notably, we illustrate the ability to execute controllable local and global shape manipulations. Our code, programs, datasets and Blender add-on are available at https://github.com/threedle/GeoCode.

Keywords: curves and surfaces, image-based modelling, modelling, modelling interfaces

CCS Concepts: • Computing methodologies → Machine learning; Mesh models; Parametric curve and surface models; • Human-centred computing → Interactive systems and tools

1. Introduction

Devising an expressive and intuitive parametrized program that generates structurally valid 3D shapes demands a high level of expertise and is a long-standing goal in computer graphics. A key challenge involves translating the user's edit intent to low-level geometric instructions that will adhere to the desired attributes while maintaining the structural validity of the shape. A promising approach for achieving control over manipulations of 3D shapes is through *procedural methods* that leverage a set of instructions to create a shape.

In this work, we present GeoCode, a new paradigm for editing complex, high-quality shapes with programs that are expressive (produce detailed shapes) and executable (enforce structural validity). Notably, GeoCode breaks away from a coarse bounding cuboid representation of shape parts. Instead, we develop rich programs that build shapes from the ground up, using sets of Béizer curves. GeoCode programs are capable of producing diverse, human-interpretable and structurally plausible shape programs that employ various rules such as curves, attachment points, bevelling, separation of structure from appearance, symmetries and more. We conducted a user study demonstrating that people without any 3D modelling experience were able to understand GeoCode's design methodology and build procedural shape programs that generate high-quality geometries using GeoCode.

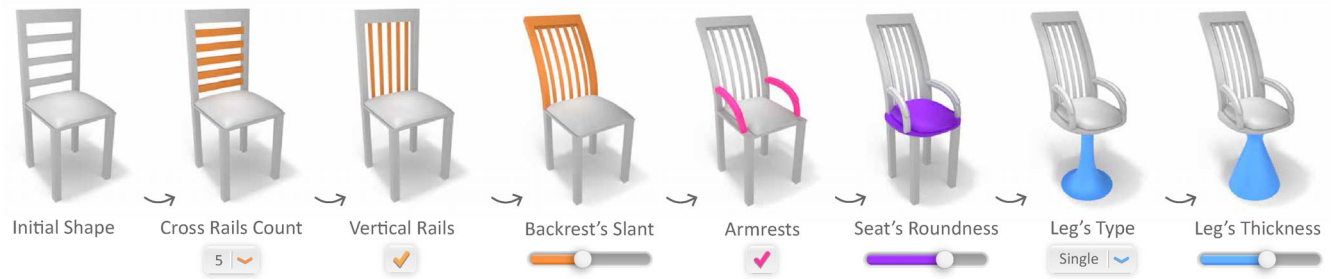

Figure 1: *Interpretable shape control. GeoCode is a new paradigm that aims to ease the process of creating complex, high-quality procedural shape programs while balancing expressiveness and ease of use. We showcase one of our shape programs, that produces detailed chair shapes while maintaining structural validity through various human-interpretable edits.*

To demonstrate the effectiveness of our procedural modelling methodology, we built three domain-specific programs. Each program is built for a particular shape domain and exposes a large human-interpretable parameter space, which keeps the shape structurally valid for every combination while capturing geometric features that are represented by the human-interpretable parameter space. Each parameter space models a wide range of geometric properties and produces a variety of detailed shapes. Shapes can be edited further in an intuitive way and without interacting with the geometry, as shown in Figure 1. Additionally, the shapes can be easily mixed and interpolated using their interpretable parameter representation and our programs propagate the changes while maintaining a valid shape (see Figure 3).

Each program is coupled with a shape recovery network, which learns to infer the program parameters from input point clouds or sketches. We leverage the effectiveness of our procedural programs to generate a large dataset for each of the domains by sweeping over their corresponding parameter space. We train on our automatically generated datasets, to learn a mapping from the input modality (sketch or point cloud) to the interpretable program parameter space, which in turn yields a structured and easily editable shape. We show that our system generalizes to inputs from different distributions than the training set, such as free-form user-created sketches, sketches generated from images in the wild, noisy point cloud data, real-world point cloud scans and more.

In summary, we propose a framework to simplify the process of building general part-based procedural programs. Our contributions are summarized in three main points:

- **Blender Add-On**: Our main contribution is allowing non-experts to create expressive procedural 3D programs of part-based shapes. To allow that, we supply a Blender-based [Com18] add-on providing users with custom nodes that follow our design methodology. We prove its effectiveness in a user study in which non-expert users built 3D programs for a predefined class (vases) but were also able to build a program for a completely novel class (ceiling lamps) using an illustration that defines the requirements and basic guidance in a timed exercise. This demonstrates that the building blocks in GeoCode are modular enough to extend to novel shape domains even by non-experts.
- **Procedural Programs**: We build three domain-specific programs (chairs, vases and tables) using our procedural modelling methodology, all of which have a large parameter space and offer great variability in the shapes they produce. Another expert program (cabinets) was developed for further discussion and the testing of GeoCode's modularity.
- **Parametric Prediction Network**: We present a system that learns to infer the parameters of the shape program from an input point cloud or sketch. Given a shape program, all the network components are automatically adjusted, from dataset generation to inference. We compare our network with a competing parametric network and present measurable improvements.

As far as we can ascertain, we are the first to develop a bottom-up curve-based framework aimed at editing shapes in a structurally valid manner. Given that there are no direct methods for comparison, we compare and evaluate our method to what alternative methods can achieve with their baselines, and demonstrate quantitative and qualitative improvements.

2. Related Work

Procedural modelling. Procedural modelling is the process of generating shapes based on given rules. The benefit of using rules to generate shapes is that the input is much more limited than the shape that is produced, and additionally, making modifications or generating variations of the shape is a much less involved process that does not require the artist to remodel the shape. Contrary to the use of *neural implicit representations* of geometry [CZ19, MON*19, PFS*19, HAESB20, ZLWT22, HPG*22] the resulting procedurally generated shapes can be integrated into any existing 3D pipeline and the artist can freely modify the shape using existing visual graphical user interfaces.

Various methods were used to generate the procedural model, an early form of procedural modelling is a grammar used to describe plants known as L-Systems [Lin68]. It was also successfully applied for buildings [MWH*06] and for cities [PM01]. Another type of grammar is *set grammar* where shapes are considered as symbols [Sti82, WWSR03]. An extension of set grammar is *shape grammar* which was first introduced in [SG71], where the basic primitives are geometries that may change by the rules (for example, when one geometry is attached to another). This method saw a lot of success in urban design as evident from works such as [MWH*06, SM15, AAAD23, LWW08, WGV*21].

Another class of procedural modelling is *model synthesis* where a 3D model is given as an input example and a larger more complex model is generated [Mer07, MM08, MM10]. Many works also provided the users with an interactive way to edit the shapes through the use of a visual system [LWW08, LD99, NPA*22, DAB16, VGDA*12, VABW09, MS09]. In this work, we explore procedural modelling through the use of visual programming and more specifically with the use of node graph systems. There have been a few works that explored such visual programming frameworks, the work [BP12] created such a framework in an attempt to simplify the development process of procedural buildings. Another attempt to create a more generalized framework through the use of a novel node graph system is the work [GK07] and notably, it also allowed some interactive editing in the viewport. Through our user study, we show that our method is intuitive even for non-experts, and with only a little guidance, our participants were able to successfully generate intricate shape programs.

Inverse procedural modelling. The second part of our work resembles inverse procedural modelling in the sense that we predict the input parameters to our procedural models given a point cloud or a sketch of the shape. Similar works include [SPK*14, RMGH15], and another work that based the prediction on a neural network [HKYM17]. The approach taken by [HKYM17] was to split the continuous and discrete parameters into two separate networks. We later use it as a baseline for our work for sketch inputs. However, the work [HKYM17] used existing procedural models [MM12] and accepted only sketches as input. In contrast, in our work, the main goal is to lower the bar for procedural model creation for non-experts. A recent work [HSIvK23] used inverse procedural modelling to aid expert programmers in iteratively crafting programs based on a given collection of reference shapes. We acknowledge another effort related to the parameter space, where auto-encoders were used to map complex parameter spaces to lower-dimensional ones to ease the process of generating shapes from procedural programs [YAMK15].

Considering our shape recovery network and programs, a single structure-aware program built using our method and visual programming can encode various structures of the shape and relations between parts from its construction. However, other works tried to solve a more difficult problem, where the relations between parts are also inferred, either as grammar or code [JBX*20, JCG*21, WYD*14, VAB10, JWR22, Mer23, JCR24, KJR24] or as graphs, often also inferring relations between the parts [LXC*17, MGY*19, WGK*22, PvGG20]. We compare our work to [JBX*20, MGY*19] solely in terms of the expressibility that our programs can achieve compared to methods that use cuboids as primitive geometry to build the shapes. There is also limited work on inferring the visual graph node for procedural materials [GHS*22, HGH*22, HGH*23]. Some works first retrieve a pre-built procedural material from a database and then optimize its parameters to match a given material [HDR19, SLH*20, TRT*22]. We also recognize other works that operate with another procedural modelling technique called constructive solid geometry such as InverseCSG [DIP*18] or PLAD [JWR22].

Shape reconstruction and editing. Several works used deep learning methods to reconstruct a 3D object from a sketch. DELANOY et al. [DAI*18] and LI et al. [LPL*18] predict voxel grids and depth with normal maps, respectively, which are then converted to meshes. Other approaches use Poisson Surface Reconstruction [KBH06, KH13] to convert a predicted point cloud to a mesh [LGK*17, YYHYZ20, ZQG*21], deform a template mesh [ZGG21], learn an implicit representation [CCR*22], or differentiable mesh representation [RLR*20, GRYF21]. In contrast, our approach directly outputs a high-quality and editable mesh.

Another approach produces a CAD shape to improve the structural integrity and editing capabilities of the reconstructed mesh. Sketch2CAD [LPBM20] and Free2CAD [LPBM22] trained neural networks to parse 2D sketches into sequences of CAD commands and ComplexGen [GLP*22] created CAD shapes from input point clouds. However, editing one CAD operation changes only a part of the shape, making the preservation of its structural integrity challenging. In our method, changing one parameter results in local changes that propagate to the rest of the shape, maintaining physical validity while keeping other geometric features intact.

We note that there is a large body of research on recovering the underlying surface mesh from a point cloud input [HDD*92, KBH06, KH13, HMGCO20, MHZ*21]. Our goal in this paper is different. We aim to produce an intuitively editable mesh version of the point cloud, while surface reconstruction works focus mainly on recovering a holistic 3D shape.

3. Method

Our main goal is to simplify the process of building procedural programs, even for complex shapes. We do this, first, by defining a procedural modelling methodology, and secondly, by providing the user with an extension to Blender's Geometry Node graph system [Fou21b] in the form of an easy-to-install add-on. Upon installation, the user will be provided with custom nodes that adhere to our procedural modelling methodology. Our second goal in this work is to evaluate the expressiveness and effectiveness of programs built using our method. Given an input object, represented as a 3D point cloud or a 2D sketch image, we want to recover the parameter set that will form a 3D shape that best matches the given input. We develop three domain-specific procedural programs parameterized by a very large human-interpretable parameter set. The 3D shape is recovered by training a neural network to infer the set of parameters that will yield the best matching 3D shape once fed to the procedural program; See Figure 2 for an overview.

3.1. Procedural modelling methodology

With the main goal of simplifying the creation of new procedural models through visual programming, we needed to balance two opposing forces. On the one hand, we want to supply non-expert users with nodes that can easily generate complex meshes with multiple input parameters. On the other hand, we want the nodes to be reusable across many shape domains. In other words, we want the nodes we provide to have a lot of expressiveness but we also want them to generalize. Our procedural modelling methodology achieves this by the use of several notions which we explain below: triplets, curve sampling, and symmetries.

Triplets. Guided by that notion, we modelled shape elements using *triplets*. A triplet is a subgraph comprised of three nodes. Firstly,

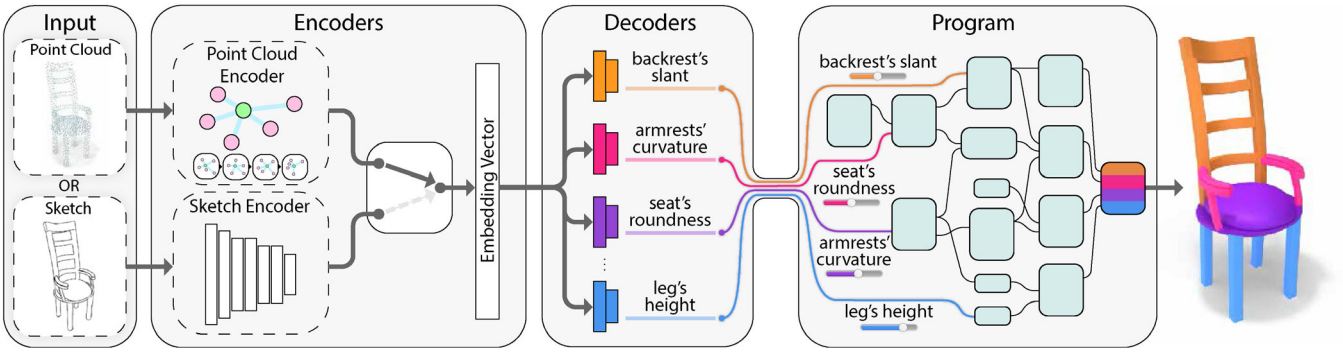

Figure 2: *System overview. GeoCode learns to map a point cloud or a sketch input to an intuitively editable parameter space. The input passes through the corresponding encoder to obtain an embedding vector which is then fed to a set of decoders that predict the interpretable parameters. The program enforces a set of rules that, given a parameter representation, produces a high-quality shape by construction.*

Figure 3: *Structural integrity. GeoCode ensures that an edited shape remains structurally plausible. Changing the parameters of the shape (left) to* ***modify the seat*** *in isolation will lead to an undesirable result (middle). Our program (right) properly propagates the edit to the remainder of the shape.*

two curves: an open curve C_1, which describes a path in the 3D space, and a planar, closed curve C_2, which describes the profile of the shape. Secondly, a style node receives the curve and the profile curve as inputs. In the simplest form of the style node, the profile curve C_2 is extruded along the curve C_1, and creates the 3D mesh.

Within the style node, additional control on the appearance of the final 3D mesh is achieved by setting the scale of the profile C_2 at each point along the curve C_1. A triplet and its links are demonstrated in Figure 7 where a leg of a chair is generated using only three nodes. Such a leg design may be controlled by 4 continuous parameters: (1) the length of the leg; (2) leg bottom thickness; (3) leg serrations; (4) the shape of the profile curve C_2 (e.g. interpolate it from a square to a circle). In our add-on design, we make use of curve editor nodes to make the process of adding more styles more intuitive. How a leg triplet might be used within a program is demonstrated in step 2 in Figure 4.

We note that on some occasions it makes sense to use mesh primitives such as a sphere or a plane. We use this, for example for the vase's lid handle (see top middle vase in Figure 6). We can also deform primitives using parameterized mathematical expressions that are applied per vertex.

Curve sampling. An artist may approach building shape programs with primitives or pre-modelled parts that are parameterized with blend shapes, a method based on shape interpolation. However, a notable benefit of cultivating curves to build shape elements is the ease of defining attachment points on a curve, by setting points with relative distances along the curve. This means, given a shape element $\mathcal{A}$, which is built from a curve C_1 and a profile curve C_2, an attachment point $\mathbf{p}_1$ on the shape element is defined by a single float number $\mathbf{p}_1 \in [0, 1]$. A value of 0.0 is the start point of the curve C_1, while a value of 1.0 is the endpoint of the curve. We attach a shape element $\mathcal{B}$ to shape element $\mathcal{A}$ by defining an attachment point on each one. We optionally set the orientation of $\mathcal{B}$ to match the normal of C_1 at the attachment point $\mathbf{p}_1$. Steps 5 and 6 in Figure 4 show how the top rail and cross-rails are attached to the frame of the chair in this manner. Scaling shape elements is achieved by calculating distances between attachment points.

Symmetries. Other structural relations do not require attachment points and rely simply on symmetry. This is shown in steps 2 and 3 in Figure 4, where we use reflective symmetry to replicate the chair's legs and frame. We also employ rotational symmetry, for example, chairs with swivel legs and vases that have multiple handles as shown in Figure 6.

Structural integrity discussion. To invoke a discussion about how structural integrity is maintained using GeoCode, let us consider Figure 4. In step 3, consider the Bézier curve C_1'' and assume we now want to make the chair narrower at the top. With the current program as described in Figure 4, we assume a constant backrest width, so our cross rails (in step 6) will not fit between a narrowing backrest. However, with a more robust design, and using GeoCode's sampling node and a repeat-zone system (loop), the correct length between the two frames at each point can be calculated and assigned to the individual cross rails. We exemplify this in Figure 5 where we take a step further and ensure that the sides of each drawer follow exactly the cabinet's profile, even if there is an error in the indexing (see the two bottom examples). This is done inside Blender's repeat zone, by using GeoCode's curve array-sampling node, Loft

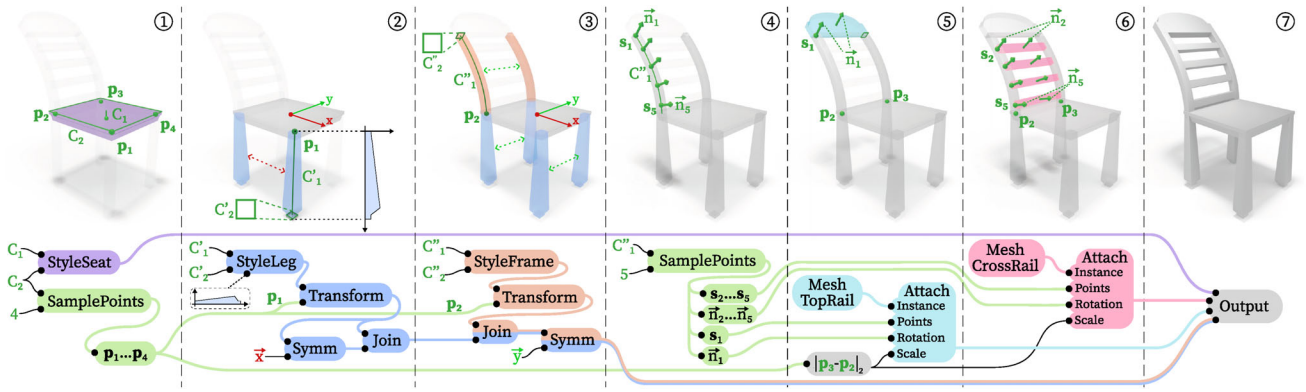

Figure 4: *Procedural shape construction. GeoCode generates a set of low-level instructions that adhere to high-level rules to produce a structurally valid shape. We show an example of using GeoCode. In step 1, curve C_1, profile curve C_2 and the node* StyleSeat *form a triplet that generates the seat. C_2 is sampled 4 times which gives us the attachment points $\mathbf{p}_1..\mathbf{p}_4$. The profile curve C_2 can parametrically deform into a circle, while the sampled points follow along with those changes. In step 2, curve C'_1 and profile curve C'_2 generate the leg's mesh when combined with* StyleLeg. *A parameterized function determines the scale of the profile along C'_1, e.g. to make the leg thicker at the bottom. The leg is then attached to point $\mathbf{p}_1$. Next, we mirror the leg using a symmetry node* Symm *along the x axis. Lastly, we* Join *the mesh of both legs and pass them further as one unit. In step 3, the frame of the backrest is generated by the triplet of Bézier curve C''_1 profile curve C''_2 and a* StyleFrame *node. The frame mesh is attached to point $\mathbf{p}_2$. We* Join *the new frame and the legs from the previous step and use a* Symm *node to replicate them to the other side of the chair. Step 4 shows how attachment points for the top-rail and cross-rails are created dynamically on C''_1 by sampling it 5× using* SamplePoints *node. The sampling node outputs the points and the normal at each point. In step 5, a top-rail shape is attached to the top of the frame at point $\mathbf{s}_1$ and is oriented to match the normal $\bar{\mathbf{n}}_1$. We scale its width to connect seamlessly to both sides of the frame. In step 6, the cross rails are attached at points $\mathbf{s}_2..\mathbf{s}_5$ with orientations matching $\bar{\mathbf{n}}_2..\bar{\mathbf{n}}_5$ respectively, and scaled to fit between both sides of the frame. We obtain the final shape (step 7) after joining the parts together.*

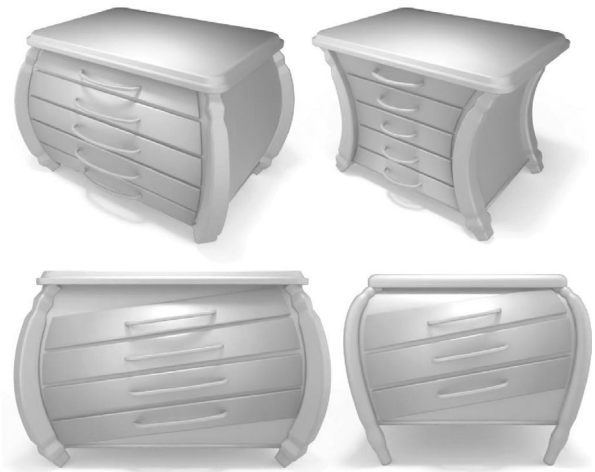

Figure 5: *Structural integrity discussion. We show that GeoCode can be used in cases where the shape elements depend on two points. The drawers in this example stay between the cabinet frames no matter the width. The handles of the drawers are also nodes in GeoCode add-on and they remain attached even when the drawers are at an angle due to a simulated indexing error.*

node and a solidify node (all are supplied in the GeoCode add-on). GeoCode also provides curves that attach at two ends such as the handle in Figure 4, which stays connected correctly even when introduced with indexing errors that cause the drawers to be misaligned. At times when structural integrity is hard to maintain by rules within the program, GeoCode offers a simple way to detect intersections between parts and rule out samples for training based on the detection.

3.2. GeoCode shape programs

We build three procedural programs based on our procedural modelling methodology. The programs are implemented in Blender's [Com18] Geometry Nodes [Fou21b] as a directed acyclic graph (DAG) comprised of *nodes* and *links*. Nodes can hold anything from a single value, all the way to complex geometry. Links can pass along anything from a single value to complex geometry that may include other attributes at the fundamental geometry level (per vertex, per edge, per face, etc.). Examples of operation nodes include math operations or vector operations; mesh primitives, such as cubes or spheres; line primitives such as a Bézier curve; and rigid transformation (translation, rotation and scale). Selected nodes are parameterized by the input parameters, allowing the user to interact with the program and control the resulting shape. The DAG culminates into a single output node which holds the final 3D shape.

Our programs support three types of human-intuitive input parameters: discrete, binary and continuous. The programs offer disentangled control over the shape, which enables modification of a specific part while keeping all others intact. It also models complex structural interactions, such that one part influences another, and the latter is adapted automatically to preserve part contact and retain the structural integrity of the shape. For example, replacing the square seat with a round one in Figure 3 makes the seat narrower, which also changes the leg's position and decreases the width of the

Figure 6: *Shape gallery. Showing reconstructed shapes on our test set. Our procedural program produces high-quality geometry given a 3D point cloud or a 2D sketch and contains consistent part segmentation information across the resulting shapes.*

backrest accordingly. The programs also hold the part segmentation information as shown in Figure 6.

The input to our programs is a set of human-interpretable parameters. These can set the structure of the shape (e.g. the height of a chair's seat or the points where the handles will be attached to a vase) or modify shape elements' appearance (e.g. the width of a leg or the roundness of the seat of a chair). Structural edits affect the open curves (i.e., C_1) and propagate to other shape elements using the attachment points and symmetries enforced by our program.

Edits to the appearance of shape elements affect the profile curves (i.e., C_2) and can cause structural edits as well. An example of this case is increasing the seat's roundness, which causes the chair to get narrower by imposing structural edits on the shape that bring the legs and the frame of the chair closer together. Considering the opposite direction, edits to the structure of the shape cannot affect the appearance of any of the shape elements. For example, changing the height of the seat of the chair will not affect the seat's roundness or the appearance of the legs.

3.3. Mapping to the program space

To map a point cloud or sketch input to the human-interpretable parameter representation, we employ an encoder-decoder network architecture. The encoder embeds the input into a global feature vector. Then, we use a set of decoders where each one translates the embedding vector to a single parameter. Together, the final interpretable representation is obtained. Finally, we run the program and recover the 3D shape. Figure 2 illustrates our system design.

Problem formulation. We formulate the shape recovery problem as predicting the human-interpretable parameters from a given point cloud or sketch. Let us denote the program parameters as $\{p_i\}$, where each parameter can take N_i discrete values. Continuous program parameters, such as thickness or height, are discretized uniformly over their range. The ground-truth value of the parameter is encoded by a one-hot vector $\mathbf{y}_i \in \{0,1\}^{N_i}$. The ground-truth representation for all the parameters is the concatenation of all $\{\mathbf{y}_i\}$, which we denote as $\mathbf{y} \in \{0,1\}^{\sum_i N_i}$.

The network prediction of the program parameters is as follows:

$$\hat{\mathbf{y}}_{\texttt{pc}} = \mathbf{D}(\mathbf{E}_{\texttt{pc}}(\mathbf{c})), \quad \hat{\mathbf{y}}_{\texttt{sketch}} = \mathbf{D}(\mathbf{E}_{\texttt{sketch}}(\mathbf{s})) \tag{1}$$

where $\hat{\mathbf{y}}_{\texttt{pc}}$ and $\hat{\mathbf{y}}_{\texttt{sketch}}$ are the predicted parameters in one-hot representation from the point cloud $\mathbf{c}$ or sketch $\mathbf{s}$, respectively, $\mathbf{E}_{\texttt{pc}}$ is the point cloud encoder and $\mathbf{E}_{\texttt{sketch}}$ is the sketch encoder, and $\mathbf{D}$ denotes the shared decoders.

To train the network, we use our program and construct a dataset of the point cloud, sketch and ground-truth triplets, i.e., $\mathcal{D} = \{(\mathbf{c}, \mathbf{s}, \mathbf{y})\}$. This process is automated given a program. Then, we train the network with the loss function:

$$\mathcal{L} = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{c},\mathbf{s},\mathbf{y})\in\mathcal{D}} \text{CE}(\hat{\mathbf{y}}_{\texttt{pc}}, \mathbf{y}) + \text{CE}(\hat{\mathbf{y}}_{\texttt{sketch}}, \mathbf{y}) \tag{2}$$

where CE denotes the *cross-entropy loss*. We employ the *part existence label* to address parameters that are not represented in the final shape (e.g. a chair with no handles). Please refer to Appendix C in the supplementary for more information.

4. Experiments

We present our user study on GeoCode, as well as qualitative and quantitative evaluations of our method's performance on shape recovery and editing. We demonstrate GeoCode's ability to recover 3D shapes from point clouds and sketches from a held-out test set from our dataset and from shapes in the wild. Furthermore, we demonstrate the editing capabilities of our system on the reconstructed shapes, such as modifying part geometry, mixing two shapes and interpolating between shapes. Finally, we present an ablation study. Please refer to the supplementary material for an additional ablation study, additional experiments and a discussion about failure cases.

Dataset and implementation details. Each of our three programs handles a different shape domain of chairs, vases and tables, and has 59, 39 and 36 human-interpretable parameters, respectively.

To train and evaluate our system, we automatically generated the train, validation and test datasets using our shape programs. Our datasets are lean compared to the vast number of parameter combinations in each shape program. For each possible choice of a single parameter, we only take 30 random shapes for the training set and three shapes for the validation and test sets. We also ensure the parameter choice is taking effect in each random shape. For each generated 3D shape, we sample 1500 points using Farthest Point

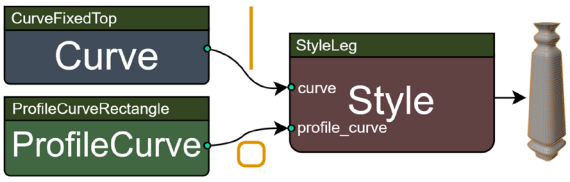

Figure 7: *Triplets. Triplets are built using three nodes. The style node accepts a curve node and a profile curve node. It sets the scale at each point along the curve based on given parameters and generates a complex mesh. Here a GeoCode triplet is used to generate a parametric leg with serrations and variable thickness.*

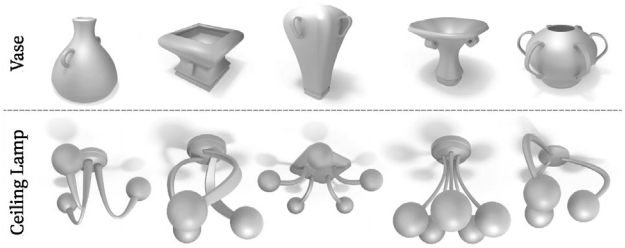

Figure 8: *User study shape programs. Examples from the programs that our participants generated. The vase program has 20 input parameters and was generated as a step-by-step tutorial which also teaches the concepts of GeoCode. The ceiling lamp program has 15 input parameters and was given as an exercise to craft the program without knowing the connections between the nodes. Both programs encode high variability and output high-quality meshes.*

Sampling [ELPZ97] and an additional 800 randomly sampled points. We render the 2D sketch images using Blender's [Com18] rendering engine *Freestyle* [Fou21a] from three camera angles while training only on two of them. Sketches are randomly augmented using horizontal flip, stroke dilation and stroke erosion.

In total, the training set for each domain contains 9570 chairs, 9330 vases and 6270 tables. The validation and test sets each contain 957 chairs, 933 vases and 627 tables. For each domain, we ensure that no two shapes in the dataset are the same.

For the point cloud encoder, we use DGCNN [WSL*19] and employ a VGG architecture [SZ15] for the sketch encoder. The decoder utilizes a multi-layer perceptron with three layers for each program parameter. Since each human-interpretable parameter may have a different number of values, the output layer size of each decoder varies according to the number of possible classes of the parameter the decoder is responsible for. Our system ensures that, given a program, the network adjusts those attributes automatically.

Blender add-on. Blender [Com18] offers a node graph system for procedural modelling called Geometry Nodes [Fou21b]. We built an add-on that extends it by providing a multitude of custom nodes that adhere to the procedural modelling methodology (see Section 3.1). For a list of the nodes we provide, please refer to Appendix A in the supplementary material. We rewrote our chair program using the GeoCode add-on and found it to massively improve the manageability of crafting large programs compared to using only Blender's vanilla nodes. Quantitatively, the program written without the add-on had 819 nodes and 1K links while the program written with the add-on had 339 nodes and 462 links. Both variants are available in the supplementary material. Our add-on includes nodes that are grouped into categories: curves - open curves of various types; profile curves - planar closed curves that are extruded along an open curve; styles - nodes that determine the scale of the profile curve on each point along the open curve; meshes - usually a pre-prepared triplet (see Figure 7); sampling - nodes that sample a given curve at single or multiple intervals; symmetries - such as mirror or radial duplication; visualization - debug nodes, for example to visualize vectors or points. Combined into a single framework, we show that these nodes allowed non-experts to craft complex procedural programs with only little guidance and in two different shape domains.

4.1. GeoCode user study

We conducted a user study to evaluate the benefits of using our procedural modelling methodology as captured by our GeoCode. The user study is comprised of two parts. In the first part, users learn about GeoCode's procedural modelling methodology and then follow a detailed guide to build a procedural program for vases (examples of shapes generated by the vase program are shown in the top row of Figure 8). The vase program has over 20 input parameters, yet it is comprised of only 10 GeoCode custom nodes and 5 built-in nodes. In the second part of our user study, the participants are asked to create a program that generates **ceiling lamps**. During the ceiling lamp exercise, we supplied the participants with a high-level design of the ceiling lamp program that explains its requirements. The required nodes are hinted but with no direct context and no node links provided, the participant never saw the node graph in advance. The participants are asked to implement what they learned during the guided vase program and put together a ceiling lamp procedural program based on the requirements. The final ceiling lamp program has 15 input parameters and is comprised of 9 GeoCode custom nodes and 5 built-in nodes. Questions were asked in four phases: before starting the user study (general knowledge), after completing the guided vase program, after completing the ceiling lamp exercise and finally, after completing the user study.

User study results. We had a total of 12 participants. Based on Likert-scale general knowledge questions, about half of our participants were familiar with Blender. However, none of them were familiar with Geometry Nodes. Eight of our participants predicted that building a vase program would take them more than a week. However, all our participants completed the guided vase program successfully and did so in an average time of 45 min. Similarly, eight participants predicted a week of work on a ceiling lamp program. However, our study shows that nine of our participants completed the exercise with no errors in an average of 32 min. Two other participants made a single mistake in the graph. The participants were able to implement GeoCode methodology and nodes to create an *entirely* new domain. This emphasizes the reusability of GeoCode custom nodes across different shape domains.

Table 1: *Increased user confidence in building procedural programs.*

Shape domain	Before	After	t-statistic	p-value
Vase	2.0	**4.7**	6.74	3.2e-5
Ceiling lamp	1.9	**4.5**	5.95	9.7e-5
Novel domain	1.6	**3.75**	5.82	1.2e-4

Note: We show the confidence of participants in our user study in building procedural models for a given shape domain before and after learning about GeoCode through our user study. Confidence scores are given on a Likert scale (1–5, where 5 is the most confident) averaged over 12 participants. Users were more confident in their abilities to build *novel* procedural programs even outside of the predefined domains.

To evaluate the effect of GeoCode on non-expert users, we asked our participants about their confidence in building such programs before and after they completed the guide. We summarize the results in Table 1. We observe a significant increase in the confidence of non-experts to craft new procedural programs.

4.2. Shape recovery

We evaluate our method's ability to recover 3D shapes from samples in our test set. We also consider out-of-distribution examples, including shapes from COSEG [WAvK*12], real-world scan-based datasets such as ScanNet [DCS*17], hand-drawn sketches and sketches generated by CLIPasso [VPB*22] from images in the wild. Our supplementary contains additional experiments where we compare our work to HUANG et al. [HKYM17], show the importance of expressive shape parts, and compare the reconstruction accuracy between point cloud and sketch inputs.

For quantitative evaluations, we use the bi-directional Chamfer Distance [BTBW77]. The squared distance from each point in one point cloud to the closest point in the other point cloud. We use 10k randomly sampled points on the ground truth and reconstructed shapes.

Results on our test set. We first verify qualitatively that our system can correctly reconstruct shapes given an input point cloud or sketch. In Figure 6, we show examples of reconstructed shapes and their corresponding inputs. We do the same for the ceiling lamp program that was created by our user study participants in Figure 9. In both, the reconstructions are visually similar to the input point clouds and sketches, confirming that our system can faithfully recover shapes for inputs within our data distribution, even for programs that were made by non-experts.

Comparison to HUANG et al. [HKYM17**]**. We test our suggested network against [HKYM17], which takes a different approach to predicting parameters that are the input to a procedural program. In their work, they train two different networks, one for predicting continuous parameters and another dedicated to predicting discrete parameters. We trained [HKYM17] on our training dataset until convergence. We implemented the loss function as depicted in their paper, and verified our actions with the authors. Then, we evaluated both methods on sketches from our test set. Table 2 shows the average Chamfer Distances on all our domains separated by whether the sketches' camera angles were trained on or not. Please refer to a qualitative comparison in Appendix B.

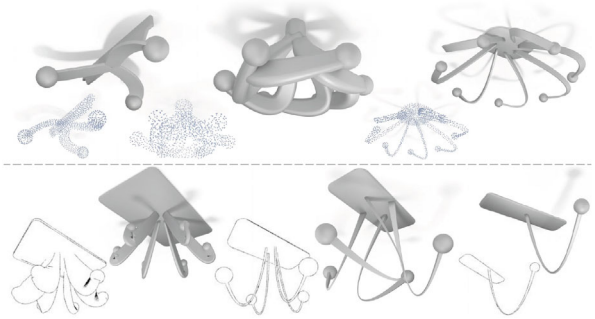

Figure 9: *Ceiling lamp test set. Showing reconstructed ceiling lamps on the test set. We trained our model using the program created by our non-expert participants during the user study, and we showcase reconstruction results from 3D point clouds and sketches.*

Table 2: *Comparison to HUANG et al. [HKYM17].*

Dataset	Method	Trained angles ↓	Novel angle ↓
Chair	HUANG et al.	2.142e-3	8.030e-3
	GeoCode	1.057e-3	8.075e-3
Vase	HUANG et al.	4.328e-3	9.271e-3
	GeoCode	4.875e-3	9.788e-3
Table	HUANG et al.	4.051e-3	9.667e-3
	GeoCode	2.497e-3	8.533e-3
Ceiling lamp	HUANG et al.	1.361e-2	2.236e-2
	GeoCode	1.403e-2	2.272e-2

Note: Comparing the average Chamfer distance on our test set to HUANG et al.[HKYM17]. We explain the result by the fact that our method is more resilient to shapes that are not rotationally symmetric, thus, the Chamfer Distance for both the chair and table domains is improved by GeoCode. In both cases, we observe a highly statistically significant result (with a p-value of 2.2e-10 and 1.8e-6 respectively). Additionally, both methods perform similarly when the sketches are drawn from a novel angle.

In Appendix B we discuss a phenomenon where the shapes recovered from the novel camera angle are typically more rounded and in this experiment, we found that the work [HKYM17] is susceptible to the same issue. However, while GeoCode only suffers from that issue for the novel angle, the method described in [HKYM17] seems to show the same issue even for trained camera angles. This explains the results shown in Table 2.

Reconstruction from out-of-distribution sketches. We conduct a qualitative evaluation of sketch inputs from AmateurSketch-3DChair and ProSketch-3DChair datasets, both datasets contain sketches of shapes in ShapeNet [CFG*15] drawn by artists from various angles as part of the work Sketch-Based 3D Shape Generation [ZQG*21]. AmateurSketch-3DChair is a dataset created by amateur artists and the ProSketch-3DChair is created by professionals. Figure 10 shows that while some discrepancies between the sketches and their reconstructions exist, overall GeoCode is able to capture key attributes from both sketch types even for shapes that are considered out-of-distribution. We report an average Chamfer Distance

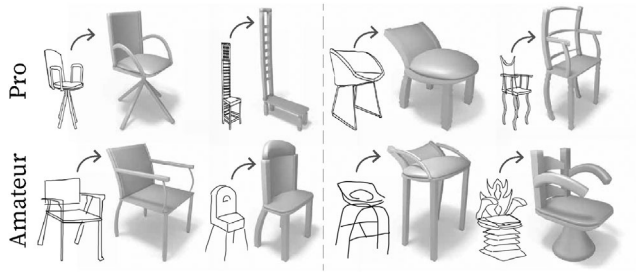

Figure 10: *Reconstruction from amateur and professional sketch datasets. We use sketches from AmateurSketch and ProSketch chairs datasets [ZQG*21] and show qualitative results for 3D shape recovery using GeoCode. On the left, we show reconstruction for samples that are within the distribution of shapes the chair program can generate; on the right, we show recovery for shapes that are out of distribution. GeoCode does incur errors such as the dimensions of the second professional sketch or the handles in the third amateur sketch. However, even when the samples are particularly challenging, GeoCode is able to capture the main geometrical attributes in the recovered shape.*

of 0.042 for the AmateurSketch dataset (1005 shapes) and 0.049 for the ProSketch dataset (500 shapes).

Qualitative comparison to HUANG et al. [HKYM17] on out-of-distribution sketches. We go on to compare GeoCode to HUANG et al. [HKYM17] on reconstruction from out-of-distribution sketches. Figure 11 shows a qualitative comparison of the same sketches used in Figure 10 while comparing both methods. Both methods experience difficulties, however, the reconstruction results of GeoCode far exceed those of HUANG et al. [HKYM17]. We direct the readers' attention to the bottom two rows where the samples are considered out-of-distribution since the programs cannot generate a shape that closely matches those samples. In these two rows, the advantage of GeoCode is even more evident.

Reconstruction from hand-drawn sketches. Figure 12 shows reconstruction examples from sketches that we drew by hand (free-form sketches), pictures from the web that we converted into sketches by outlining the shapes (sketch outlining), and sketches from the web (sketches in the wild). Our system is able to generalize and capture the main features from the sketches and produce shapes that are visually similar to the input sketches. These sketches exhibit different styles and are drawn from different angles compared to the sketches we trained on.

Reconstruction from CLIPasso sketches. Figure 13 shows reconstruction examples from sketches generated by CLIPasso [VPB*22]. We use CLIPasso to convert images of chairs, vases and tables found in the wild to sketches with various numbers of strokes. The sketches produced by CLIPasso are noisy and often contain artefacts, these make them visually different from sketches found in our datasets. We observe that, overall, the shapes reconstructed from our system are sensible and correctly recover features from the shapes in the original images. This suggests that our network is able to express shapes with human-interpretable parameters given an out-of-distribution input.

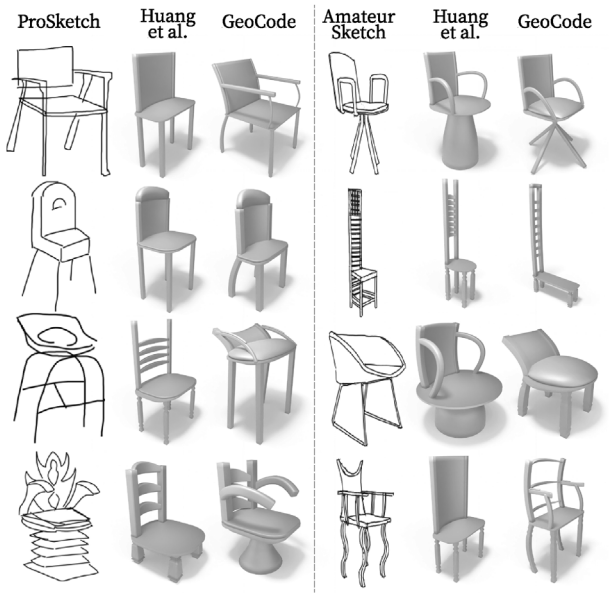

Figure 11: *Qualitative comparison to HUANG et al. [HKYM17] on out-of-distribution sketches. We use AmateurSketch and ProSketch sketches to show a qualitative analysis of 3D shape recovery between the two methods. The top two rows are samples that we consider in-distribution, while the bottom two rows are samples that are out-of-distribution. We find that GeoCode is more capable of capturing important geometric details.*

Reconstruction from out-of-distribution point clouds. To further demonstrate the generalization capability of GeoCode, we choose three known 3D shape datasets and reconstruct shapes from these datasets for point cloud inputs. First, we consider two real-world scan-based datasets, ScanObjectNN [UPH*19] and ScanNet [DCS*17], which include LiDAR scans of objects and scenes, respectively. For ScanObjectNN, we use the point clouds from the main dataset. For ScanNet, we extract objects from the scene using the provided segmentation masks and randomly sample point clouds from them. We also consider ShapeNet [CFG*15] and produce point clouds by randomly sampling its shapes. For all the datasets, we end up with point clouds of 2048 points.

In Figure 14 we show the reconstruction for two domains, chairs and tables. Our method is able to recover shapes that capture the main geometric features of the out-of-distribution samples. Moreover, point clouds produced from scanned objects are contaminated by noise and partiality. Still, GeoCode is able to produce compelling results in these cases as well.

4.3. Shape editing

Beyond shape recovery, we show that our interpretable parameter space allows for easy shape editing. This property is exemplified by shape mixing and shape interpolation.

Shape mixing. We show that our system can mix shapes by selecting a set of parameters from the human-interpretable parameter space representation of a source shape and copying them to a target

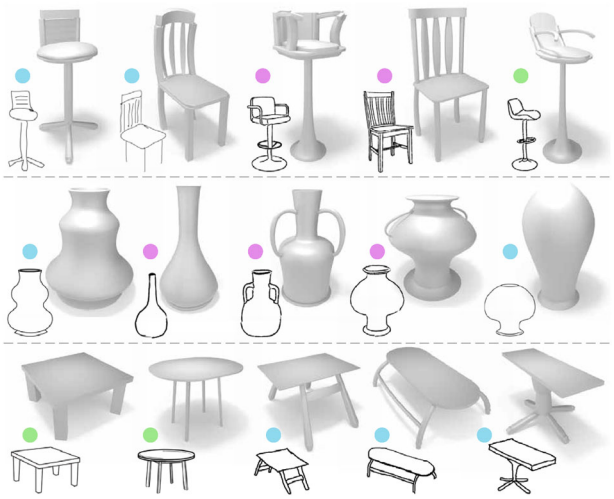

Figure 12: *Hand-drawn sketches. We show the ability of GeoCode to recover an editable 3D shape from free-form sketches, sketch outlining and sketches in the wild. Our system produces a 3D shape that captures the main attributes of the input sketches.*

Figure 13: *Sketches from images. Starting with an input image, we automatically create a sketch using CLIPasso [VPB*22]. Our method generates editable shapes that closely match the original images and the corresponding sketches that have a variety of styles.*

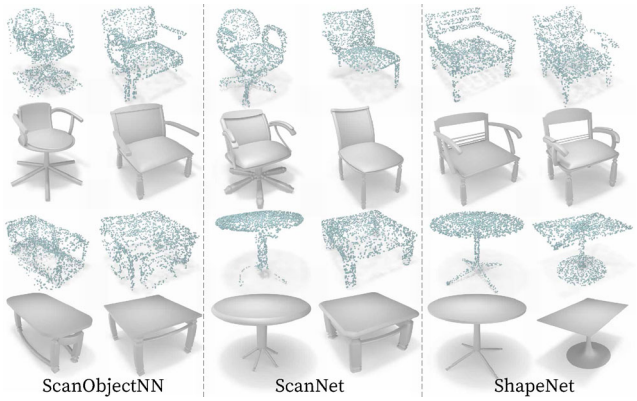

Figure 14: *Generalization to out-of-distribution point clouds. We show reconstruction results of GeoCode for point clouds from various datasets. We use scanned objects from ScanObjectNN [UPH*19] and ScanNet [DCS*17] and also test on point clouds from ShapeNet [CFG*15]. Our method outputs plausible reconstructions, despite hindered point cloud quality that is characterized by noise and missing parts.*

Figure 15: *Shape mixing. We demonstrate shape mixing by taking geometric features from a source shape and copying them to a target shape. GeoCode will either add the selected geometric features to the target shape (e.g. handles are added to the second vase), or it will override them in the target shape (e.g. cross rails in the backrest of the first chair are replaced with vertical rails). In both cases, the structural integrity of the shape is maintained.*

shape. Figure 15 shows mixing examples between pairs of shapes reconstructed from point clouds and sketches. The resulting shapes are based on the first shape with the addition of selected parts from a second shape. The final mixed shape is structurally valid and physically plausible.

Shape interpolation. GeoCode is effective in interpolating between shapes. Figure 16 shows interpolation between pairs of shapes reconstructed from point clouds and sketches where we interpolate across all the parameters for $\alpha \in [0, 1]$ in 0.2 increments. In our supplementary, we demonstrate interpolation over selected set geometric features.

4.4. Ablation study

We perform an ablation study to support our choice of using classification for float values. Please refer to Appendix B for an additional ablation study.

Figure 16: *Shape interpolation. We uniformly interpolate between two (left and right) GeoCode shapes. Interpolations contain gradual changes (continuous parameters) as well as sharp structural changes (discrete or binary parameters).*

Table 3: *Classification vs. regression.*

Method	Point cloud ↓	Sketch ↓
Classification	**0.00040**	**0.00106**
Regression	0.00198	0.00364

Note: We train the chair network in two different ways. Classification: continuous parameters are discretized and the loss function is cross-entropy, as described in Section 3. Regression: we employ cross-entropy loss for discrete and binary parameters, but we use regression to predict continuous ones. We can see that the Chamfer Distance is better overall when using classification loss for all the parameters. The results are highly statistically significant, with p-values of 1.9e-24 and 7.2e-50 for point cloud and sketch reconstruction respectively.

Classification vs. regression. In our work, we supported discrete, Boolean and continuous parameters. We explained in Section 3 that continuous parameters are discretized and that the loss function is cross-entropy. In this ablation experiment, we test the reconstruction in terms of average Chamfer Distance while comparing the use of classification vs. regression for the continuous parameters.

In Table 3 we can see that using classification for all the parameters yields better average Chamfer Distance results. In Figure 17 we show two examples from the chair domain that show the difference between the two methods. Another important part of this experiment is explained in Appendix C where we discuss the *part existence label* and its role in this experiment.

4.5. Case study with an expert

We discussed our work with a computer graphics artist with over 10 years of experience in the industry and approximately one year of experience with Blender Geometry Nodes. Firstly, the expert confirmed that, to their knowledge, they are 'not aware of any other framework that resembles what GeoCode add-on does'.

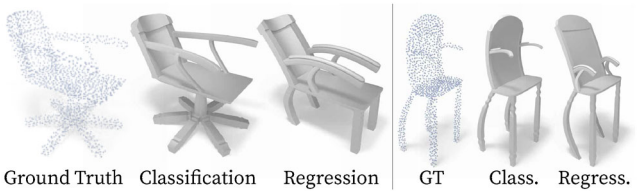

Figure 17: *Classification vs. regression. Qualitative comparison between reconstruction when continuous parameters are predicted using regression vs. discretizing them and predicting them using classification. We show the improved reconstruction on selected chair samples favouring prediction using classification.*

The expert correctly noted that 'Non-experts will need at the very least a short introduction to Blender and Geometry Nodes before jumping into using GeoCode add-on'. This is the very reason that our user study (refer to Section 4.1) required an introduction to Blender and our add-on to provide basic tools to our participants. They proceeded to say, 'Experts would probably find it extremely logical and intuitive. GeoCode itself is quite easy to use for a person who knows their way around the Geometry Nodes system'.

When asked about part-based shapes, and taking the Cabinet example (see Figure 5), the expert said 'Given the task of creating such a complex program, I would probably find it unmanageable without using a framework such as GeoCode'.

We also asked about what improvements they thought could be made to GeoCode within the confines of Blender. The answers revolved around pain points that many CG artists deal with on a daily basis - optimizing the topology for game assets, UV unwrapping and selectable preset materials. All of which can form future improvements to our work.

Finally, the expert stated that 'Experts can even use GeoCode add-on and programs as a resource for learning Geometry Nodes'.

5. Limitations

Building the chair, vase and table programs which have 59, 39 and 36 parameters took roughly 2 weeks, a single week and 5 days, respectively. While this is a significant amount of time, these programs were crafted with the concepts of GeoCodein mind, but before we created the Blender add-on. With our Blender add-on, we were able to re-program the chair program in a mere 4 days. The ceiling lamps program (19 parameters) from our user study was built by non-expert participants in an average time of 32 min after approximately 40 min of a guided tutorial. In addition, to further strengthen the trust in GeoCode's ability to extend to new domains we created the cabinet program (25 parameters) in a record time of just 2 days.

GeoCode does not guarantee the structural integrity of the shapes simply by using the add-on and the methodology that we presented in this work. However, a good design will lead to valid shapes even when introduced with mistakes, we examine this using our cabinet program when discussing the structural integrity in 3.1. In addition to providing useful nodes in their own right, we also handle many pitfalls that programmers may encounter when using vanilla

Geometry Nodes, such as unexpected curve direction flips, alignment with axes, attachment to rotated elements and much more.

We also recognize that GeoCode will not be useful for every conceivable shape, for example, we would not recommend it for generating organic shapes such as faces or some animals. It is intended to be used for part-based shapes.

Finally, while our network's predictions from point clouds are impressive, our predictions from sketches are only improving upon the existing methods in certain conditions, and interestingly, the predictions from the novel angle suffer from the tendency for rotational symmetry as we explained in Appendix B.

6. Conclusion

In this paper, we presented GeoCode, a novel method that aims to lower the barrier for crafting shape programs that can generate 3D shapes using a human-interpretable parameter space. We show the effectiveness of shape programs by building three domain-specific procedural programs controlled by intuitive parameter spaces and training a neural network to predict the parameter representation for an input point cloud or sketch. We showed that our system produces structurally valid 3D geometry and enables editing of the resulting shape easily and intuitively. Our user study demonstrated that our framework enables users with no prior 3D modelling experience to produce shape programs. Specifically, users can take the core components from these programs and build a completely novel class (ceiling lamps), which we did not create a program for.

In the future, we are interested in incorporating additional interactive features, such as viewport manipulations and edits to shapes. Incorporation of additional edit modalities (such as drawing a part on a screen) combined with our visual programs may provide additional opportunities for program exploration and design.

Acknowledgements

We thank the University of Chicago for providing the AI cluster resources, services, and the professional support of the technical staff. This work was also supported in part by gifts from Adobe Research. Finally, we would like to thank R. Kenny Jones, Chen Dudai, Noam Sahar, and the members of 3DL for their thorough and insightful feedback on our work.

Open access funding was provided by the University of Chicago.

References

[AAAD23] Alfadalat M. A., Al-Azhari W., Dabbour L.: Procedural modeling based shape grammar as a key to generating digital architectural heritage. *ACM Journal on Computing and Cultural Heritage 16*, 4 (2023), 1–17.

[BP12] Barroso S., Patow G.: Visual language generalization for procedural modeling of buildings. In *CEIG* (2012), pp. 57–66.

[BTBW77] Barrow H. G., Tenenbaum J. M., Bolles R. C., Wolf H. C.: Parametric correspondence and Chamfer matching: Two new techniques for image matching. In *IJCAI* (1977).

[CCR*22] Cheng Z., Chai M., Ren J., Lee H.-Y., Olszewski K., Huang Z., Maji S., Tulyakov S.: Cross-modal 3D shape generation and manipulation. In *European Conference on Computer Vision (ECCV)* (2022), pp. 303–321.

[CFG*15] Chang A. X., Funkhouser T., Guibas L., Hanrahan P., Huang Q., Li Z., Savarese S., Savva M., Song S., Su H., Xiao J., Yi L., Yu F.: *ShapeNet: An Information-Rich 3D Model Repository*. arXiv preprint arXiv:1512.03012 (2015).

[Com18] Blender Foundation.: *Blender - A 3D Modelling and Rendering Package*. Blender Foundation. 2018. http://www.blender.org. (Accessed 24 July 2022).

[CZ19] Chen Z., Zhang H.: Learning implicit fields for generative shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 5939–5948.

[DAB16] Demir I., Aliaga D. G., Benes B.: Proceduralization for editing 3D architectural models. In *2016 Fourth International Conference on 3D Vision (3DV)* (2016), pp. 194–202.

[DAI*18] Delanoy J., Aubry M., Isola P., Efros A. A., Bousseau A.: 3D sketching using multi-view deep volumetric prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 1 (2018), 1–22.

[DCS*17] Dai A., Chang A. X., Savva M., Halber M., Funkhouser T., Nießner M.: ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 5828–5839.

[DIP*18] Du T., Inala J. P., Pu Y., Spielberg A., Schulz A., Rus D., Solar-Lezama A., Matusik W.: InverseCSG: Automatic conversion of 3D models to CSG trees. *ACM Transactions on Graphics 37* (2018), 1–16.

[ELPZ97] Eldar Y., Lindenbaum M., Porat M., Zeevi Y. Y.: The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing 6*, 9 (1997): 1305–1315.

[Fou21a] Blender Foundation.: *Freestyle Introduction*. Blender Foundation. 2021. https://docs.blender.org/manual/en/latest/render/freestyle/introduction.html. (Accessed 24 July 2022).

[Fou21b] Blender Foundation.: *Geometry nodes Introduction*. Blender Foundation. 2021. https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/introduction.html. (Accessed 06 November 2022).

[GHS*22] Guerrero P., Hašan M., Sunkavalli K., Měch R., Boubekeur T., Mitra N. J.: MatFormer: A generative model for procedural materials. *ACM Transactions on Graphics 41*, 4 (2022), 1–12.

[GK07] Ganster B., Klein R.: An integrated framework for procedural modeling. In *Proceedings of the 23rd Spring Conference on Computer Graphics* (2007), pp. 123–130.

[GLP*22] Guo H., Liu S., Pan H., Liu Y., Tong X., Guo B.: ComplexGen: CAD reconstruction by B-Rep chain complex generation. *ACM Transactions on Graphics 41*, 4 (2022), 1–18.

[GRYF21] Guillard B., Remelli E., Yvernay P., Fua P.: Sketch2Mesh: Reconstructing and editing 3D shapes from sketches. In *International Conference on Computer Vision* (2021), pp. 13023–13032.

[HAESB20] Hao Z., Averbuch-Elor H., Snavely N., Belongie S.: DualSDF: Semantic shape manipulation using a two-level representation. In *IEEE Conference on Computer Vision and Pattern Recognition* (2020), pp. 7631–7641.

[HDD*92] Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W.: Surface reconstruction from unorganized points. *ACM SIGGRAPH Computer Graphics 26*, 2 (1992), 71–78.

[HDR19] Hu Y., Dorsey J., Rushmeier H.: A novel framework for inverse procedural texture modeling. *ACM Transactions on Graphics 38*, 6 (2019), 1–14.

[HGH*22] Hu Y., Guerrero P., Hasan M., Rushmeier H., Deschaintre V.: Node graph optimization using differentiable proxies. In *ACM SIGGRAPH Conference Proceedings* (2022), pp. 1–9.

[HGH*23] Hu Y., Guerrero P., Hasan M., Rushmeier H., Deschaintre V.: Generating procedural materials from text or image prompts. In *ACM SIGGRAPH 2023 Conference Proceedings* (2023), pp. 1–11.

[HKYM17] Huang H., Kalogerakis E., Yumer E., Mech R.: Shape synthesis from sketches via procedural models and convolutional networks. *IEEE Transactions on Visualization and Computer Graphics 23*, 8 (2017), 2003–2013.

[HMGCO20] Hanocka R., Metzer G., Giryes R., Cohen-Or D.: Point2Mesh: a self-prior for deformable meshes. *ACM Transactions on Graphics 39*, 4 (2020), 126-1.

[HPG*22] Hertz A., Perel O., Giryes R., Sorkine-Hornung O., Cohen-Or D.: SPAGHETTI: Editing implicit shapes through part aware generation. *ACM Transactions on Graphics 41*, 4 (2022), 1–20.

[HSIvK23] Hossain I., Shen I.-C., Igarashi T., van Kaick O.: Data-guided authoring of procedural models of shapes. *Computer Graphics Forum 42*, 7 (2023), e14935.

[JBX*20] Jones R. K., Barton T., Xu X., Wang K., Jiang E., Guerrero P., Mitra N. J., Ritchie D.: ShapeAssembly: Learning to generate programs for 3D shape structure synthesis. *ACM Transactions on Graphics 39*, 6 (2020), 1–20.

[JCG*21] Jones R. K., Charatan D., Guerrero P., Mitra N. J., Ritchie D.: ShapeMOD: Macro operation discovery for 3D shape programs. *ACM Transactions on Graphics 40*, 4 (2021), 1–16.

[JCR24] Jones R. K., Chaudhuri S., Ritchie D.: Learning to infer generative template programs for visual concepts. In *Forty-first International Conference on Machine Learning* (2024).

[JWR22] Jones R. K., Walke H., Ritchie D.: PLAD: Learning to infer shape programs with pseudo-labels and approximate distributions. In *IEEE Conference on Computer Vision and Pattern Recognition* (2022).

[KBH06] Kazhdan M., Bolitho M., Hoppe H.: Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing 7*, 4 (2006).

[KH13] Kazhdan M., Hoppe H.: Screened Poisson surface reconstruction. *ACM Transactions on Graphics 32*, 3 (2013), 1–13.

[KJR24] Kapur S., Jenner E., Russell S.: Diffusion on syntax trees for program synthesis. *arXiv preprint arXiv:2405.20519* (2024).

[LD99] Lintermann B., Deussen O.: Interactive modeling of plants. *IEEE Computer Graphics and Applications 19*, 1 (1999), 56–65.

[LGK*17] Lun Z., Gadelha M., Kalogerakis E., Maji S., Wang R.: 3D shape reconstruction from sketches via multi-view convolutional networks. In *International Conference on 3D Vision* (2017), pp. 67–77.

[Lin68] Lindenmayer A.: Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology 18*, 3 (1968), 280–299.

[LPBM20] Li C., Pan H., Bousseau A., Mitra N. J.: Sketch2CAD: Sequential CAD modeling by sketching in context. *ACM Transactions on Graphics 39*, 6 (2020), 1–14.

[LPBM22] Li C., Pan H., Bousseau A., Mitra N. J.: Free2CAD: parsing freehand drawings into CAD commands. *ACM Transactions on Graphics 41*, 4 (2022), 1–16.

[LPL*18] Li C., Pan H., Liu Y., Tong X., Sheffer A., Wang W.: Robust flow-guided neural prediction for sketch-based freeform surface modeling. *ACM Transactions on Graphics 37*, 6 (2018), 1–12.

[LWW08] Lipp M., Wonka P., Wimmer M.: Interactive visual editing of grammars for procedural architecture. In *ACM SIGGRAPH 2008 papers*. (2008), pp. 1–10.

[LXC*17] Li J., Xu K., Chaudhuri S., Yumer E., Zhang H., Guibas L.: GRASS: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics 36*, 4 (2017), 1–14.

[Mer07] Merrell P.: Example-based model synthesis. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (2007), pp. 105–112.

[Mer23] Merrell P.: Example-based procedural modeling using graph grammars. *ACM Transactions on Graphics (TOG) 42*, 4 (2023), 1–16.

[MGY*19] Mo K., Guerrero P., Yi L., Su H., Wonka P., Mitra N., Guibas L.: StructureNet: Hierarchical graph networks for 3D

shape generation. *ACM Transactions on Graphics 38*, 6 (2019), 1–19.

[MHZ*21] Metzer G., Hanocka R., Zorin D., Giryes R., Panozzo D., Cohen-Or D.: Orienting point clouds with dipole propagation. *ACM Transactions on Graphics 40*, 4 (2021), 1–14.

[MM08] Merrell P., Manocha D.: Continuous model synthesis. In *ACM SIGGRAPH Asia 2008 Papers*. (2008), pp. 1–7.

[MM10] Merrell P., Manocha D.: Model synthesis: A general procedural modeling algorithm. *IEEE Transactions on Visualization and Computer Graphics 17*, 6 (2010), 715–728.

[MM12] Mech R., Miller G.: The deco framework for interactive procedural modeling. *Journal of Computer Graphics Techniques (JCGT) 1*, 1 (2012), 43–99.

[MON*19] Mescheder L., Oechsle M., Niemeyer M., Nowozin S., Geiger A.: Occupancy networks: Learning 3D reconstruction in function space. In *IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 4460–4470.

[MS08] McCrae J. P., Singh K.: Sketch-Based Path Design. In *Proceedings of Graphics Interface* (2009), pp. 95–102.

[MWH*06] Müller P., Wonka P., Haegler S., Ulmer A., Van Gool L.: Procedural modeling of buildings. *ACM Transactions on Graphics 25*, 3 (2006), pp. 614–623.

[NPA*22] Niese T., Pirk S., Albrecht M., Benes B., Deussen O.: Procedural urban forestry. *ACM Transactions on Graphics (TOG) 41*, 2 (2022), 1–18.

[PFS*19] Park J. J., Florence P., Straub J., Newcombe R., Lovegrove S.: DeepSDF: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 165–174.

[PM01] Parish Y. I., Müller P.: Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), pp. 301–308.

[PvGG20] Paschalidou D., van Gool L., Geiger A.: Learning unsupervised hierarchical part decomposition of 3D objects from a single RGB image. In *IEEE Conference on Computer Vision and Pattern Recognition* (2020), pp. 1060–1070.

[RLR*20] Remelli E., Lukoianov A., Richter S., Guillard B., Bagautdinov T., Baque P., Fua P.: MeshSDF: Differentiable iso-surface extraction. In *Advances in Neural Information Processing Systems 33* (2020), 22468–22478.

[RMGH15] Ritchie D., Mildenhall B., Goodman N. D., Hanrahan P.: Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 1–11.

[SG71] Stiny G., Gips J.: Shape grammars and the generative specification of painting and sculpture. In *IFIP congress (2) 2*, 3 (1971), pp. 125–135.

[SLH*20] Shi L., Li B., Hašan M., Sunkavalli K., Boubekeur T., Mech R., Matusik W.: MATch: Differentiable material graphs for procedural material capture. *ACM Transactions on Graphics (TOG) 39*, 6 (2020), 1–15.

[SM15] Schwarz M., Müller P.: Advanced procedural modeling of architecture. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 1–12.

[SPK*14] Stava O., Pirk S., Kratt J., Chen B., Měch R., Deussen O., Benes B.: Inverse procedural modelling of trees. *Computer Graphics Forum 33*, 6 (2014), 118-131.

[Sti82] Stiny G.: Spatial relations and grammars. *Environment and Planning B: Planning and Design 9*, 1 (1982), 113–114.

[SZ15] Simonyan K., Zisserman A.: Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations* (2015).

[TRT*22] Tchapmi L. P., Ray T., Tchapmi M., Shen B., Martin-Martin R., Savarese S.: Generating procedural 3D materials from images using neural networks. In *Proceedings of the 2022 4th International Conference on Image, Video and Signal Processing* (2022), pp. 32–40.

[UPH*19] Uy M. A., Pham Q.-H., Hua B.-S., Nguyen D. T., Yeung S.-K.: Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *International Conference on Computer Vision* (2019), pp. 1588–1597.

[VAB10] Vanegas C. A., Aliaga D. G., Benes B.: Building reconstruction using manhattan-world grammars. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010), pp. 358–365.

[VABW09] Vanegas C. A., Aliaga D. G., Benes B., Waddell P. A.: Interactive design of urban spaces using geometrical and behavioral modeling. *ACM Transactions on Graphics (TOG) 28*, 5 (2009), 1–10.

[VGDA*12] Vanegas C. A., Garcia-Dorado I., Aliaga D. G., Benes B., Waddell P.: Inverse design of urban procedural models. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 1–11.

[VPB*22] Vinker Y., Pajouheshgar E., Bo J. Y., Bachmann R. C., Bermano A. H., Cohen-Or D., Zamir A., Shamir A.: CLIPasso: Semantically-aware object sketching. *ACM Transactions on Graphics 41*, 4 (2022), 1–11.

[WAvK*12] Wang Y., Asafi S., van Kaick O., Zhang H., Cohen-Or D., Chen B.: Active co-analysis of a set of shapes. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 1–10.

[WGK*22] Wang K., Guerrero P., Kim V. G., Chaudhuri S., Sung M., Ritchie D.: The shape part slot machine: Contact-based reasoning for generating 3D shapes from parts. In *European Conference on Computer Vision (ECCV)* (2022), pp. 610–626.

[WGV*21] Willis A. R., Ganesh P., Volle K., Zhang J., Brink K.: Volumetric procedural models for shape representation. *Graphics and Visual Computing 4* (2021), 200018.

[WSL*19] Wang Y., Sun Y., Liu Z., Sarma S. E., Bronstein M. M., Solomon J. M.: Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics 38* (2019), 1–12.

[WWSR03] Wonka P., Wimmer M., Sillion F., Ribarsky W.: Instant architecture. *ACM Transactions on Graphics (TOG) 22* (2003), 669–677.

[WYD*14] Wu F., Yan D.-M., Dong W., Zhang X., Wonka P.: Inverse procedural modeling of facade layouts. *ACM Transactions on Graphics 33*, 4 (2014), 121.

[YAMK15] Yumer M. E., Asente P., Mech R., Kara L. B.: Procedural modeling using autoencoder networks. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (2015), pp. 109–118.

[YYHYZ20] Yue Z., Yulia G., Honggang Z., Yi-Zhe S.: Deep sketch-based modeling: Tips and tricks. In *International Conference on 3D Vision* (2020), pp. 543–552.

[ZGG21] Zhang S.-H., Guo Y.-C., Gu Q.-W.: In *IEEE Conference on Computer Vision and Pattern Recognition* (2021), pp. 6012–6021.

[ZLWT22] Zheng X., Liu Y., Wang P., Tong X.: SDF-StyleGAN: Implicit SDF-based StyleGAN for 3D shape generation. *Computer Graphics Forum 41*, 5 (2022), 52–63.

[ZQG*21] Zhong Y., Qi Y., Gryaditskaya Y., Zhang H., Song Y.-Z.: Towards practical sketch-based 3D shape generation: The role of professional sketches. *IEEE Transactions on Circuits and Systems for Video Technology 31*, 9 (2021), 3518–3528.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supplementary material

Videos

User study

Anonymous code and data

Blender add-on