SkexGen: Autoregressive Generation of CAD Construction Sequences with Disentangled Codebooks

Xiang Xu [1] Karl D.D. Willis [2] Joseph G. Lambourne [2]
Chin-Yi Cheng [2] Pradeep Kumar Jayaraman [2] Yasutaka Furukawa [1]

Abstract

We present SkexGen, a novel autoregressive generative model for computer-aided design (CAD) construction sequences containing sketch-and-extrude modeling operations. Our model utilizes distinct Transformer architectures to encode topological, geometric, and extrusion variations of construction sequences into disentangled codebooks. Autoregressive Transformer decoders generate CAD construction sequences sharing certain properties specified by the codebook vectors. Extensive experiments demonstrate that our disentangled codebook representation generates diverse and high-quality CAD models, enhances user control, and enables efficient exploration of the design space. The code is available at `https://samxuxiang.github.io/skexgen`.

1. Introduction

Professional designers generate a diverse set of computer aided design (CAD) models with topological or geometric variations while achieving a design goal. In early concept design, designers explore shapes with aesthetic and functional advantages. In mechanical design, designers optimize the physical properties of a part to maximize strength while minimizing weight and manufacturing cost.

Training a computational agent with the capabilities of a professional designer is an extremely challenging machine learning task. Designers traditionally rely on parametric CAD models that allow one to correct small mistakes (Camba et al., 2016), optimize geometric properties (Grasel et al., 2004), and generate a product family by altering just a few parameters (Chakrabarti et al., 2011; Maher et al., 1996). However, a parametric CAD model

---

[1]Simon Fraser University, Canada [2]Autodesk Research. Correspondence to: Xiang Xu <xuxiangx@sfu.ca>.

is brittle and fails over large design changes such as topological modifications. Furthermore, the construction of a parametric CAD model itself requires the expertise of a professional designer.

With the advance of deep learning, neural networks empower a family of exciting new methodologies (Wu et al., 2021; Willis et al., 2021a; Ganin et al., 2021) towards an intelligent system capable of diverse generation while understanding a design goal and allowing user control. This paper pushes the frontier of the state-of-the-art by introducing "SkexGen", a **sk**etch-and-**ex**trude **gen**erative model for CAD construction sequences that enhances variations in design generation while enabling effective control and exploration of the design space.

SkexGen is a novel autoregressive generative model that uses discrete codebooks (van den Oord et al., 2017) for CAD model generation. We employ a sketch-and-extrude modeling language to describe CAD construction sequences, where a sketch operation creates 2D primitives and an extrude operation lifts and combines them into 3D. Transformer encoders learn a disentangled latent representation as three codebooks, capturing topological, geometric, and extrusion variations. Given codebook vectors, autoregressive Transformer decoders generate sketch-and-extrude construction sequences, which are processed into a CAD model.

We evaluate SkexGen on a large-scale sketch-and-extrude dataset (Wu et al., 2021). Qualitative and quantitative evaluations against multiple baselines and state-of-the-art methods demonstrate that our method generates more realistic and diverse CAD models, while allowing effective control and efficient exploration of the design space not possible with prior approaches. We make the following contributions.

- SkexGen architecture that autoregressively generates high quality and diverse CAD construction sequences.

- Disentangled codebooks, which encode topological, geometric, and extrusion variations of construction sequences, enabling effective control and exploration of designs.

- Extensive qualitative and quantitative evaluations on public benchmarks, demonstrating state-of-the-art performance.

2. Related Work

Constructive Solid Geometry: 3D shapes can be expressed as a constructive solid geometry (CSG) tree, where parametric primitives, such as cuboids, spheres, and cones, are combined together with Boolean operations. This lightweight representation has been used extensively for reconstruction tasks in combination with program synthesis (Du et al., 2018; Nandi et al., 2017; 2018), neurally-guided program synthesis (Sharma et al., 2018; Ellis et al., 2019; Tian et al., 2019), unsupervised learning (Kania et al., 2020), and with specialized parametric primitives (Chen et al., 2020; Yu et al., 2022). Although CSG is a convenient representation, parametric CAD remains the most prevalent paradigm for mechanical design and makes extensive use of the sketch-and-extrude modeling operations instead.

Construction Sequence Generation: A precursor to sketch-and-extrude construction sequence generation is PolyGen (Nash et al., 2020), where n-gon mesh vertices and faces are predicted using Transformers (Vaswani et al., 2017) and pointer networks (Vinyals et al., 2015). Large-scale datasets of parametric CAD construction sequences (Seff et al., 2020; Willis et al., 2021b) have opened the door to learning directly from the modeling operations by CAD users. (Willis et al., 2021b) and (Xu et al., 2021) predict the sequence of extrude operations for partial recovery of the construction sequence without the underlying sketch information. Predicting sequences of sketch primitives (e.g., line, arc, circle) is a critical building block that forms the 2D basis of CAD and can be readily extended to 3D with the addition of the extrude operation.

The Transformer architecture has enabled sketch/sketch-and-extrude construction sequence generation in recent (Willis et al., 2021a; Wu et al., 2021) and concurrent (Para et al., 2021; Seff et al., 2022; Ganin et al., 2021) work. Although these approaches can produce a diverse range of shapes, providing user control over an existing design is more elusive. To be useful for real world CAD applications, the designer needs a way to influence the generated shape. One approach is to condition the network on user-provided images (Ganin et al., 2021), point clouds (Wu et al., 2021), or hand-drawn sketches (Seff et al., 2022). However, this approach simply converts an existing design into a CAD construction sequence representation. Instead we present a novel approach for exploring the space of related designs with separate control over topology and geometry.

Codebook Architectures: Codebooks have proven effective on a number of image and audio generation tasks since their introduction by (van den Oord et al., 2017), improving the diversity of generated images (Razavi et al., 2019) and providing additional user control (Esser et al., 2021). They are particularly suited to encoding CAD modeling

sequences due to their high structural regularity.

3. Sketch-and-Extrude Construction Sequence

We define a sketch-and-extrude construction sequence representation as a hierarchy of primitives, building on Turtle-Gen (Willis et al., 2021a) and DeepCAD (Wu et al., 2021) with several modifications to make the representation more expressive and amenable for learning (See Figure 1).

Primitive Hierarchy: A "curve" (i.e., line, arc, or circle) is the lowest-level primitive. A "loop" is a closed path, consisting of one (i.e., circle) or multiple curves (e.g., line-arc-line). A "face" is a 2D area bounded by loops, which is new in our representation. Precisely, a face is defined by one outer loop and some number of inner loops as holes, a convention in many CAD systems (Lee & Lee, 2001). A "sketch" is formed by one or multiple faces. An "extruded-sketch" is a 3D volume, formed by extruding a sketch. A "sketch-and-extrude" model is formed by multiple extruded-sketches via Boolean operations (i.e., intersection, union, and subtraction). Note that the DeepCAD (Wu et al., 2021) representation does not have a face primitive and cannot represent a sketch with multiple faces (e.g., ES_1 in Figure 1).

Construction Sequence: Following DeepCAD (Wu et al., 2021), we represent a sketch-and-extrude CAD model using a sequence with five types of tokens: 1) A topology token indicates a curve type (line/arc/circle); 2) A geometry token contains a point coordinate; 3) An end-primitive token indicates the end of a primitive (curve/loop/face/sketch/extruded-sketch); 4) An extrusion token contains parameters associated with the extrusion and Boolean operations; and 5) An end-sequence token indicates the end of the sequence. Figure 1 illustrates a sample primitive hierarchy and a sequence. We provide the full language specification in Appendix A.

4. SkexGen Architecture

SkexGen is an autoregressive generative model that learns variations of sketch-and-extrude models with three disentangled codebooks in two network branches. Figure 2 visualizes the SkexGen architecture. The "Sketch" branch learns topological and geometric variations of 2D sketches, while the "Extrude" branch learns variations of 3D extrusions (e.g., directions). Both branches are similar and this section explains the sketch branch with two encoders and one decoder. Details of the extrude branch are in Appendix B.

4.1. Topology Encoder

The topology encoder takes a subsequence of the input, where the token is either 1) a topology token (T), which indicates one of the 3 curve types (line/arc/circle); 2) an
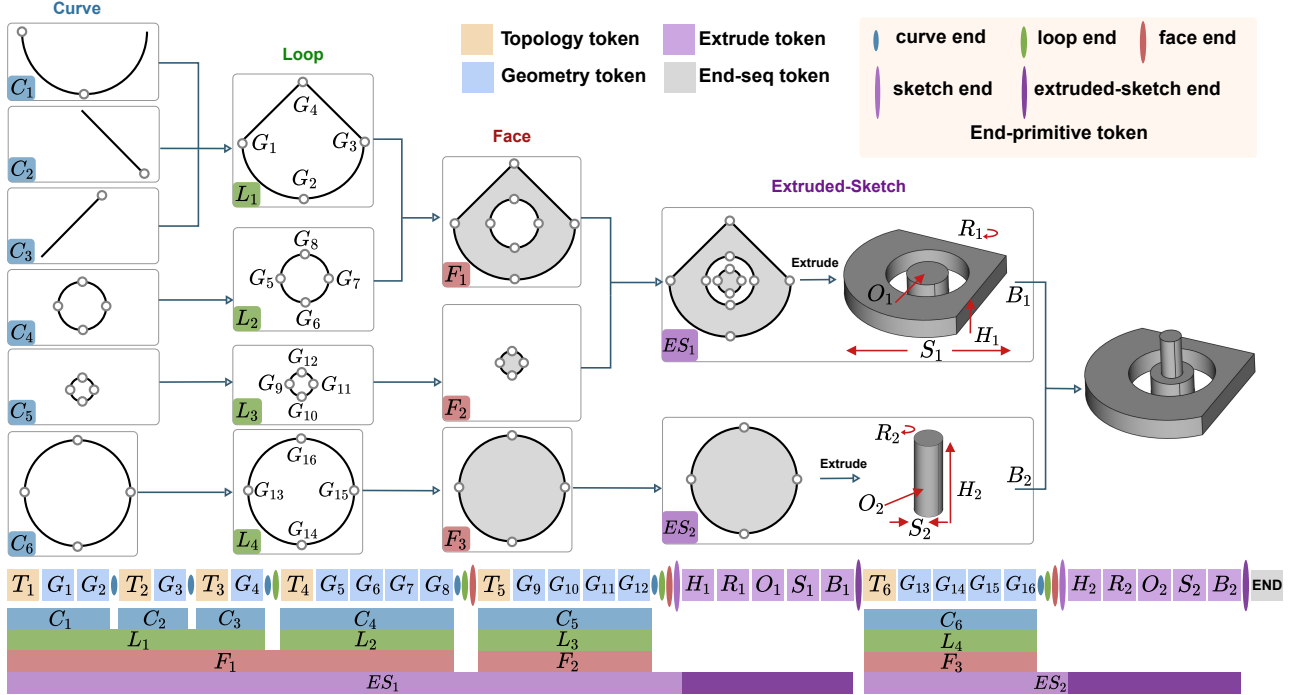
Figure 1. A sample sketch-and-extrude model is illustrated as a primitive hierarchy and a code sequence. It consists of two sketches, which are formed by faces, loops, and curves. The sequence starts by a topology token (T_1), indicating the start of a curve (type arc). Two geometry tokens follow (G_1 and G_2), each of which stores a 2D point coordinate. Note that a circle is defined by four points with four geometry tokens (e.g., G_5–G_8). An arc is defined by three points (e.g., G_1–G_3) but with two tokens, where the third point is specified by the next curve (or the first curve when a loop is closed). Similarly, a line is defined by start point (e.g., G_4). G_8 is the last point for curve (C_4), loop (L_2), and face (F_1). Therefore, the geometry token (G_8) is followed by end-primitive tokens for curve, loop, and face.

end-primitive token (E) for one of the 3 primitive types (loop/face/sketch); or 3) An end-sequence token (End), indicating the end of the sequence. Accordingly, a token is initialized with a one-hot vector of dimension $7(= 3+3+1)$.

Embeddings: The one-hot vector is transformed into a $d_E = 256$ dimensional embedding. We consider a topology token T in the subsequence where h_T^{tp} is the 7-dim one hot vector and i_T denotes its index in the input subsequence. Its embedding vector is computed as:

$$T \leftarrow \mathbf{W^{tp}} h_T^{tp} + \mathbf{p}^{(i_T)}, \tag{1}$$

where $\mathbf{W^{tp}} \in \mathbb{R}^{d_E \times 7}$ denotes a learnable matrix. $\mathbf{p}^{(i_T)} \in \mathbb{R}^{d_E}$ denotes the learnable position encoding at index i_T of the topology subsequence[1].

Architecture: The encoder is based on a standard Transformer architecture (Vaswani et al., 2017) with four layer blocks, each of which contains a self-attention layer with eight heads, layer-normalization and feed-forward layers. Following Vision Transformer (Dosovitskiy et al., 2021), the input topology information is encoded into a "code-

---

[1]With abuse of notation, T denotes a sequence token and its latent embedding. A bold-font indicates learnable parameters.

token", which is prepended to the input and initialized with a learnable embedding Z_{tp}. Let Z_{tp}^e be the embedding of a code-token at the output of the encoder. The embedding Z_{tp}^e is quantized to the closest code in the codebook of size N: $\{\mathbf{b}_{tp}^{(i)} \mid i = 1, 2 \cdots N\}$. The final code-token Z_{tp}^Q after encoding and quantization is then passed to the decoder:

$$Z_{tp}^Q \leftarrow \mathbf{b}_{tp}^{(k)}, \text{ where } k = \operatorname{argmin}_j \|Z_{tp}^e - \mathbf{b}_{tp}^{(j)}\|^2. \tag{2}$$

Here we assumed one code-token for simplicity. In practice, the topology encoder has four code-tokens and makes four output codes $\left(Z_{tp}^{Q(1)}, Z_{tp}^{Q(2)}, Z_{tp}^{Q(3)}, Z_{tp}^{Q(4)}\right)$. We tried different codebook sizes and found that $N = 500$ achieves good results.

4.2. Geometry Encoder

The geometry encoder takes a subsequence of the input, where a token is either 1) a geometry token (G); 2) an end-primitive token (E) for one of the 4 primitive types (curve/face/loop/sketch); or 3) an end-sequence token (End). The geometry token G specifies a 2D point coordinate along a curve. Since a coordinate is numerical, we discretize sketches into 64×64 (6 bits) pixels and consider
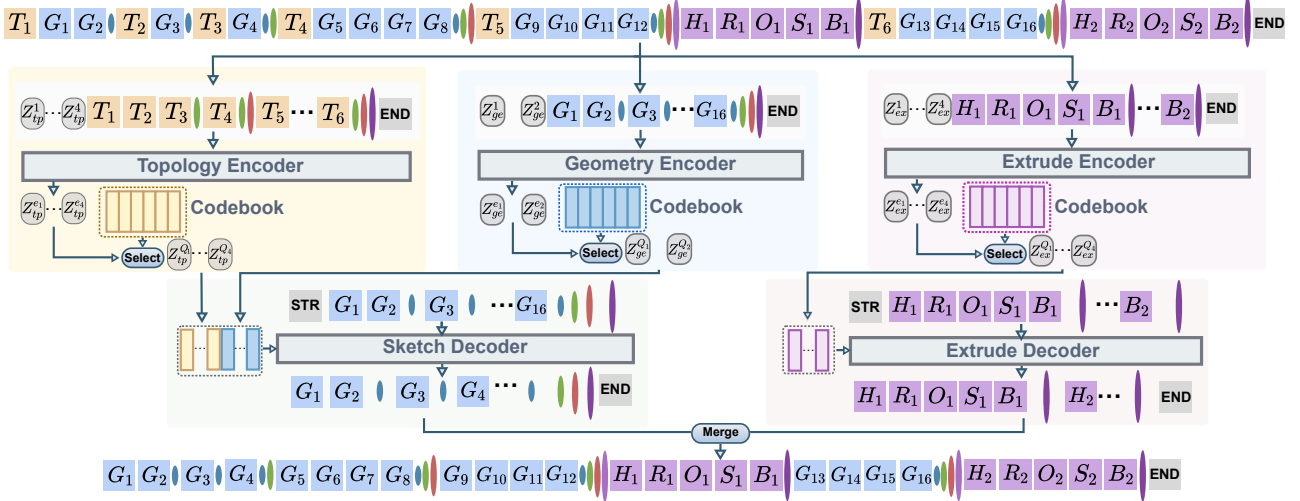
Figure 2. The SkexGen architecture has two branches. The sketch branch (left) has two encoders that learn topological and geometric variations of sketches in two codebooks. An autoregressive decoder generates the sketch subsequence given codebook vectors. The extrude branch (right) has an encoder and a decoder that learns variations of extrude and Boolean operations. The two branches are trained independently. Another autoregressive decoder learns to select effective combinations of codes from the three codebooks (not shown).

64^2 possible pixel locations[2]. Therefore, a one-hot vector of dimension $4101(= 64^2 + 4 + 1)$ uniquely determines the token information.

Embeddings: We follow Eq. 1 and use a learnable matrix $\mathbf{W^{ge}} \in \mathbb{R}^{d_E \times 4101}$ together with position encoding to initialize input token embeddings. Token E and End are initialized similarly to the topology token T, by multiplying their one-hot vector with $\mathbf{W^{ge}}$ and adding the position encoding. Geometry tokens G are initialized differently as:

$$G \quad \leftarrow \quad \mathbf{W^{ge}} h_G^{ge} + \mathbf{W^x} h_G^x + \mathbf{W^y} h_G^y + \mathbf{p}^{(i_G)}. \quad (3)$$

$h_G^{ge} \in \mathbb{R}^{4101}$ denotes the one-hot vector. The geometry tokens G have additional coordinate embeddings where $h_G^x, h_G^y \in \mathbb{R}^{64}$ is a one-hot vector indicating the x, y coordinate of the pixel. $\mathbf{W^x}, \mathbf{W^y} \in \mathbb{R}^{d_E \times 64}$ are learnable coordinate matrices. Coordinate embeddings are optional but further improve results in our experiments.

Architecture: Similar to the topology encoder, code-tokens in the geometry encoder produces the embedding $\{Z_{ge}^{e(i)}\}$ and the quantized code $\{Z_{ge}^{Q(i)}\}$. We use two code-tokens for the geometry encoder ($i = 1, 2$). Codebook size is 1000.

4.3. Sketch Decoder

The sketch decoder takes as input the topology and geometry codebooks and generates geometry tokens G and end-primitive tokens E (for curve/loop/face/sketch) to recover the sketch subsequence. Note that topology tokens T are

---

[2]6 bits yield enough precision for most CAD models as described in Vitruvion (Seff et al., 2022). More bits bring little improvement at the cost of significantly more network parameters.

not generated, as they can be inferred from the number of geometry tokens within each curve (i.e., line/arc/circle have 1/2/4 G tokens). This means that geometry encoder and sketch decoder have similar subsequence (see Figure 2).

Input: Given the past $k - 1$ tokens, the autoregressive decoder predicts the conditional probability of the kth token. The training input sequences are shifted one to the right, with the "start" symbol (initialized by the position encoding) added at front. Since the types of possible tokens in the decoder are equivalent to those of the geometry encoder, we use the same one-hot encoding scheme of dimension 4101 and also a learnable matrix of size $d_E \times 4101$ with position encoding to initialize the embedding vectors.

Output: The decoder produces a subsequence "shifted one to the left", that is, predicting the original k tokens in the input (See Figure 2). Let K be a token in the output of sketch decoder, which has a d_E dimensional embedding. We use a learnable matrix $\mathbf{W^{out}} \in \mathbb{R}^{4101 \times d_E}$ to predict the probability likelihood over 4101 classes as:

$$h_K^{\text{out}} \leftarrow \text{softmax}\left(\mathbf{W^{out}} K\right). \quad (4)$$

Cross-attention: The Transformer architecture takes four and two quantized codebook vectors from the topology and the geometry codebooks via cross-attention. To distinguish two different codebooks, we borrow the idea of position encoding and add learnable embedding vectors $\mathbf{p}^{(q_{tp})} \in \mathbb{R}^{4 \times d_E}$ and $\mathbf{p}^{(q_{ge})} \in \mathbb{R}^{2 \times d_E}$ to the topology code $\{Z_{tp}^{Q(i)}\}$ and geometry code $\{Z_{ge}^{Q(i)}\}$ respectively:

$$Z_{tp}^{Q(i)} + \mathbf{p}^{(q_{tp})} \quad \text{or} \quad Z_{ge}^{Q(i)} + \mathbf{p}^{(q_{ge})}. \quad (5)$$

The base network setting is the same as the encoder (i.e., 4 layer-blocks with 8 heads), except that it is autoregressive with masking (only previous tokens are attended).

4.4. Training

The topology encoder, geometry encoder, and sketch decoder are jointly trained with three loss functions:

$$\sum_K \text{CrossEntropy}(h_K^{\text{out}}, h_K^{\text{gt}}) + \tag{6}$$

$$\left|\left| sg\left(Z_{tp}^e\right) - \mathbf{b}_{tp} \right|\right|_2^2 + \beta \left|\left| Z_{tp}^e - sg\left(\mathbf{b}_{tp}\right) \right|\right|_2^2 +$$

$$\left|\left| sg\left(Z_{ge}^e\right) - \mathbf{b}_{ge} \right|\right|_2^2 + \beta \left|\left| Z_{ge}^e - sg\left(\mathbf{b}_{ge}\right) \right|\right|_2^2.$$

The first line computes the sequence reconstruction loss, where h_K^{out} is the predicted probability likelihood from the sketch decoder and h_K^{gt} is the ground-truth one-hot vector. The second and the third lines are standard codebook and commitment losses used with VQ-VAE (van den Oord et al., 2017)[3]. sg denotes the stop-gradient operation, which is the identity function in the forward pass but blocks gradients in the backward pass. β scales the commitment loss and is set to 0.25. This ensures the encoder output commits to one code vector. We omit explicitly writing out the multiple code-tokens within each encoder for simplicity.

Given a ground-truth subsequence, we run the two encoders and autoregressively run the decoder until the same number of tokens are generated. Instead of feeding back the generated tokens, we do teacher forcing and pass the ground-truth to the input of the decoder, which effectively allows us to train the decoder for a single step each.

4.5. Generation

SkexGen generates CAD models in two steps: 1) code generation from the three codebooks; and 2) sketch-and-extrude construction sequence generation given the codes.

Code generation: Unlike a VAE, our quantized codes do not follow a Normal distribution. After using the trained topology, geometry, and extrude encoders to obtain codes from each training sample, we train a Transformer decoder that generates codes, i.e. selecting code indexes from each of the three codebooks. Our framework also allows conditional code generation with a minor modification to the architecture. For example, a "topology conditioned code selector" selects the compatible geometry and extrude codes given topology codes. We provide full details in Appendix C.

Sequence generation: Given the codes, the sketch and the extrude decoders autoregressively generate construction subsequences separately by nucleus sampling (Holtzman et al.,

2020). The subsequences are merged to form a complete sketch-and-extrude sequence, which is then parsed to boundary representation (Weiler, 1986) using CAD software.

5. Experiments

In this section we perform experiments to understand: 1) the ability of SkexGen to generate high quality and diverse results, 2) the level of control that codebooks enable over the generation process, and 3) the performance of SkexGen for applications such as design exploration and interpolation.

5.1. Experiment Setup

Dataset: We use the DeepCAD dataset (Wu et al., 2021) which contains 178,238 sequences and a data split of 90% train, 5% validation, and 5% test. Duplicates are removed in a similar manner to Willis et al. (2021a). We separately remove duplicate sketch subsequences and duplicate extrude subsequences. Invalid sketch-and-extrude operations are also removed. The resulting training dataset contains 74,584 sketch subsequences and 86,417 extrude subsequences. For experiments with single sketches, we extract sketches from all steps of the CAD construction sequences. This leads to 114,985 training samples after duplicate removal.

Implementation details: SkexGen is implemented in PyTorch (Paszke et al., 2019) and trained on a RTX A5000. For fair comparison, we follow DeepCAD (Wu et al., 2021) and use a Transformer with four-layer blocks where each block contains eight attention heads, pre-layer normalization, and feed-forward dimension of 512. The input embedding dimension is 256. Dropout rate is 0.1. During training, we use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 0.001. Linear warm-up and gradient clipping are used as in DeepCAD. We skip code quantization in the initial 25 epochs and find this stabilizes the codebook training. For data augmentation, we add small random noise to the coordinates of the geometry tokens. SkexGen is trained for a total of 500 epochs with a batch size of 128. Maximum length is 200 for sketch subsequences and 100 for extrude subsequences. At test time, we use nucleus sampling (Holtzman et al., 2020) with top-p of 0.95 and temperature of 1.0 to autoregressively sample from the decoders.

Metrics: We use the following metrics for the quantitative evaluation. For 2D sketches, "Fréchet inception distance (FID)" (Heusel et al., 2017) measures the generation fidelity. It compares mean and covariance of real and generated data distributions. Following (Das et al., 2020), we use features from ResNet-18 (He et al., 2016) pre-trained on a human sketch classification task (Eitz et al., 2012). For 3D CAD models, "Coverage" (COV) is the percentage of real data that match generated data based on the closest

---

[3]We use the exponential moving average updates of a decay rate 0.99, which was effective in VQ-VAE-2 (Razavi et al., 2019).
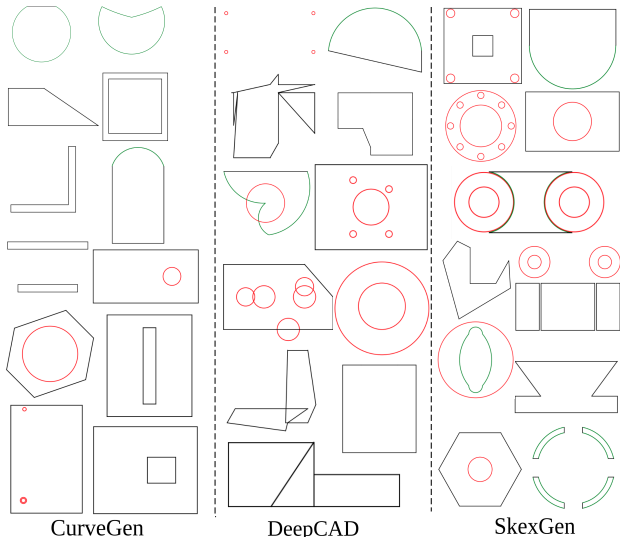
Figure 3. Uniformly sampled random sketch generation results by CurveGen, DeepCAD, and SkexGen. Circle is red, arc is green, and line is black. Visually, SkexGen generates more realistic and complex shapes.

Table 1. Quantitative evaluations on the sketch generation task. We report the FID score to measure generation fidelity and the percentage of Unique and Novel.

	FID ↓	Unique % ↑	Novel % ↑
CurveGen	109.12	93.05	83.82
DeepCAD	75.47	98.79	97.45
SkexGen	18.56	96.02	83.54

Chamfer distance of 2,000 uniformly sampled points on the surface. "Minimum Matching Distance" (MMD) is the average minimum matching distance between a generated sample and its nearest neighbor in the real set. "Jensen-Shannon Divergence" (JSD) is the similarity between the real and generated distributions based on the marginal point distribution. For both sketches and CAD models, the "Novel" score is the percentage of generated data that does not appear in the training set, the "Unique" score is the percentage of data that only appears once in the generated samples. We consider two data samples equal if all tokens in the sequence are the same after 6-bit quantization. Please refer to (Achlioptas et al., 2018; Wu et al., 2021) for details.

5.2. Random Generation

To assess the ability of SkexGen to generate high quality and diverse results, we compare 10,000 randomly generated samples from SkexGen with four other baselines: Curve-Gen (Willis et al., 2021a), DeepCAD (Wu et al., 2021), SkexGen with a single codebook without code disentanglement, and SkexGen with VAE without code quantization or code disentanglement. Other sketch generative models from

Table 2. Quantitative evaluations on the CAD generation task based on the Coverage (COV) percentage, Minimum Matching Distance (MMD), Jensen-Shannon Divergence (JSD) and the percentage of Unique and Novel. The first row is the existing state-of-the-art DeepCAD (Wu et al., 2021). The last three rows provide an ablation study of SkexGen over two features: code disentanglement (F_d) and code quantization (F_q). The second from last row disables the code disentanglement (i.e., using a single codebook), resembling standard VQ-VAE. The third from last row further disables the code quantization, resembling standard VAE.

	COV % ↑	MMD ↓	JSD ↓	Unique % ↑	Novel % ↑
DeepCAD	76.8	1.68	2.01	91.0	87.0
SkexGen - (F_d, F_q)	74.3	1.54	0.92	97.8	91.9
SkexGen - (F_d)	80.4	1.55	1.12	99.8	99.3
SkexGen	83.6	1.48	0.81	99.9	99.8

concurrent work (Para et al., 2021; Seff et al., 2022; Ganin et al., 2021) rely on sketch constraint labels, and ideally a sketch constraint solver, and are not directly comparable.

Sketch generation: We report quantitative results for sketch generation in Table 1. SkexGen has by far the best FID score. The unique and novel percentage are similar or better than CurveGen, with the exception that novel percentage is lower than DeepCAD. Closer examination of the qualitative results in Figure 3 reveals that DeepCAD produces many invalid results containing self-intersecting curves and non-watertight geometry. Since invalid results do not appear in the training data, they are all counted as novel and the score is high. The FID metric detects that the invalid data is far from the ground truth distribution and consequently the DeepCAD FID score is much worse. The Unique metric for DeepCAD is only slightly better than SkexGen; we suspect this is due to the increased noise in the generated data.

Overall, we find that sketches from SkexGen are better in terms of quality with more complex shapes, fewer self-intersections, and stronger symmetry. CurveGen also generates good quality results, but with fewer complex arrangements of rectangles and circles. DeepCAD can produce more complex shapes than CurveGen but with a lot of noise. Additional visualizations of our generated sketches are available in Appendix D.

CAD generation: Table 2 provides quantitative evaluations on the sketch-and-extrude CAD model generation. We find that SkexGen performs the best across all metrics. Qualitative results for SkexGen and DeepCAD are shown in Figure 4 with additional results available in Appendix D. SkexGen generates CAD models that are considerably more complex, exhibit symmetries, and make frequent use of the *arc* curve type, reminiscent of human design. Results from SkexGen also contain frequent multi-step sketch-and-extrude sequences, whereas DeepCAD results are mostly
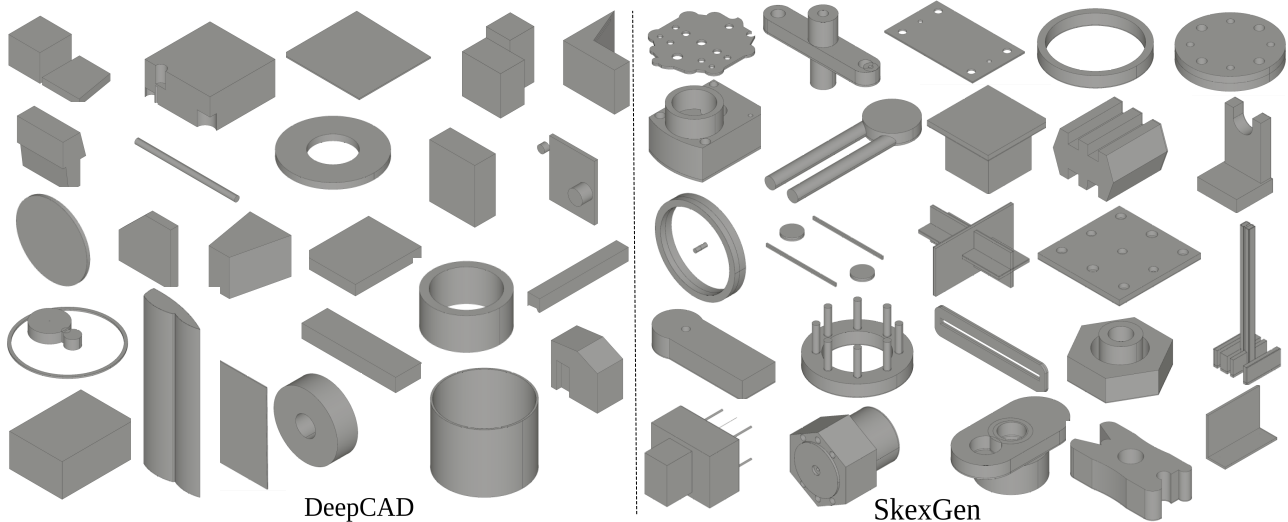
Figure 4. Uniformly sampled random sketch-and-extrude model generation results by DeepCAD and SkexGen. Visually, SkexGen generates more realistic and complex models.

single-step. The amount of similar shapes generated by DeepCAD is also high. The middle two rows in Table 2 demonstrate the effectiveness of multiple disentangled codebooks. The generation quality decreases after reduction of multiple codebooks to just one and SkexGen is similar to a VQ-VAE. Results are the worst when no codebook is used and SkexGen effectively becomes a VAE.

Running time: SkexGen is slower than DeepCAD due to autoregressive sampling, taking 90 secs to generate 10,000 samples compared to 15 secs for DeepCAD. However, it is $2\times$ faster than CurveGen since CurveGen has two dependent autoregressive decoders.

5.3. Controllable Generation

Disentangled codebooks allow effective control and design exploration. The left of Figure 5 illustrates "topology conditioned code selection" results, where the conditioned topology codes are the same for each row and the other codes are obtained by nucleus sampling. For example, in row b, all sketches contain two separate faces, which consist of an inner circle and an outer loop of six lines. The geometric properties such as face size and distance between the two faces vary. The right of Figure 5 similarly shows "geometry conditioned code selection" results, where the geometry codes are fixed for each row. In row g, all the sketches have roughly the same size and arrangement of curves, while the topology of the outer loops are different, from lines (left) to circle (middle) to arcs (right). For CAD models in the bottom rows, extrude operations further vary, influencing the 3D layout of the models (e.g., row e). Figure 6 has more controllable generation results with different conditional

code selectors, demonstrating the capability of SkexGen.

To quantitatively measure the disentanglement between the three codebooks, we follow the evaluation in β-VAE (Higgins et al., 2017). A pair of sketch-and-extrude sequences are generated by the decoders by keeping one of the topology, geometry or extrude tokens the same, and sampling the rest. A small Transformer-based classifier is trained to identify which code is fixed using the average pairwise difference in the encoded latent space over all pairs of data. The classification accuracy for SkexGen is $99.8 \pm 0.1\%$.

5.4. Applications

We demonstrate two applications enabled by the proposed SkexGen system.

Interpolation allows intuitive design exploration. Given a pair of models, we follow the same encoding and decoding process in Sect. 4, except that we linearly interpolate their codes before quantization ($\{Z^e\}$) and produce one model. Precisely, we 1) use the encoders to extract 10 codes (4 topology $\{Z_{tp}^e\}$, 2 geometry $\{Z_{ge}^e\}$, and 4 extrude $\{Z_{ex}^e\}$) from each model; 2) linearly interpolate them; and 3) perform the code quantization and autoregressive sequence generation to produce an interpolated model. Note that we take the token of the highest score during the autoregressive generation for consistent interpolation instead of the nucleus sampling. Figure 7 shows interesting topological and geometric changes over the interpolation. In column b, the topology of the sketch changes from a set of 12 lines to a set of 4 lines and 2 circles, and finally to a set of 6 circles. The radius of the middle circle gradually increases until enclosing the four smaller ones. In column h, a rect-
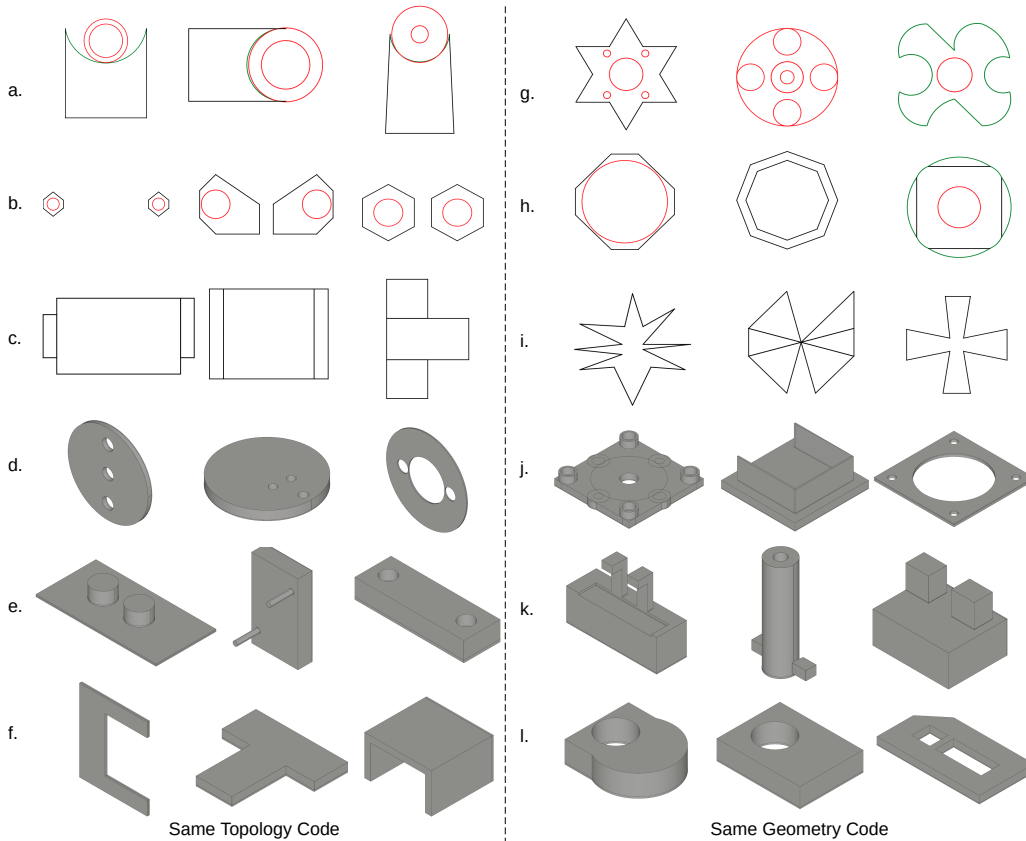
Figure 5. Controllable generation results. Each row shares the same topology (left) or the same geometry (right) codes. Other codes are randomly sampled by the code selector. Top rows are sketches and bottom rows are sketch-and-extrude CAD models.
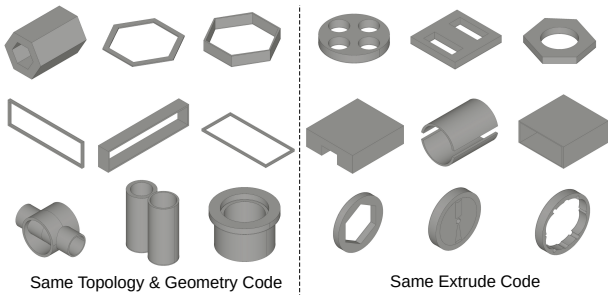

Figure 6. Controllable generation results. Each row shares the same topology and geometry (left) or the same extrude (right) codes. Other codes are randomly sampled by the code selector.

angular solid gradually morphs into a circular hollow disk. Note that the results are often not "smooth", but our interpolation tasks are particularly challenging, requiring many topological changes that are discrete in nature.

Topology, geometry, and extrude code mixing enables generation results not seen in previous work. In Figure 8 we mix the topology, geometry, or extrude codes from one data with those of another to produce a hybrid result. For example, in row a, the generated sketch has two inner circles

as in the reference topology while the placement of the two circles resembles that of the reference geometry. In row k, the reference topology indicates many cylinders and the reference geometry indicates a square, resulting in many cylinders in the arrangement of a square.

6. Conclusion

We introduced SkexGen, a novel generative model for CAD construction sequences that enhances topological and geometric variations to enable better design control and exploration. SkexGen is a step towards an intelligent system capable of generating diverse CAD models while comprehending design goals and allowing user control.
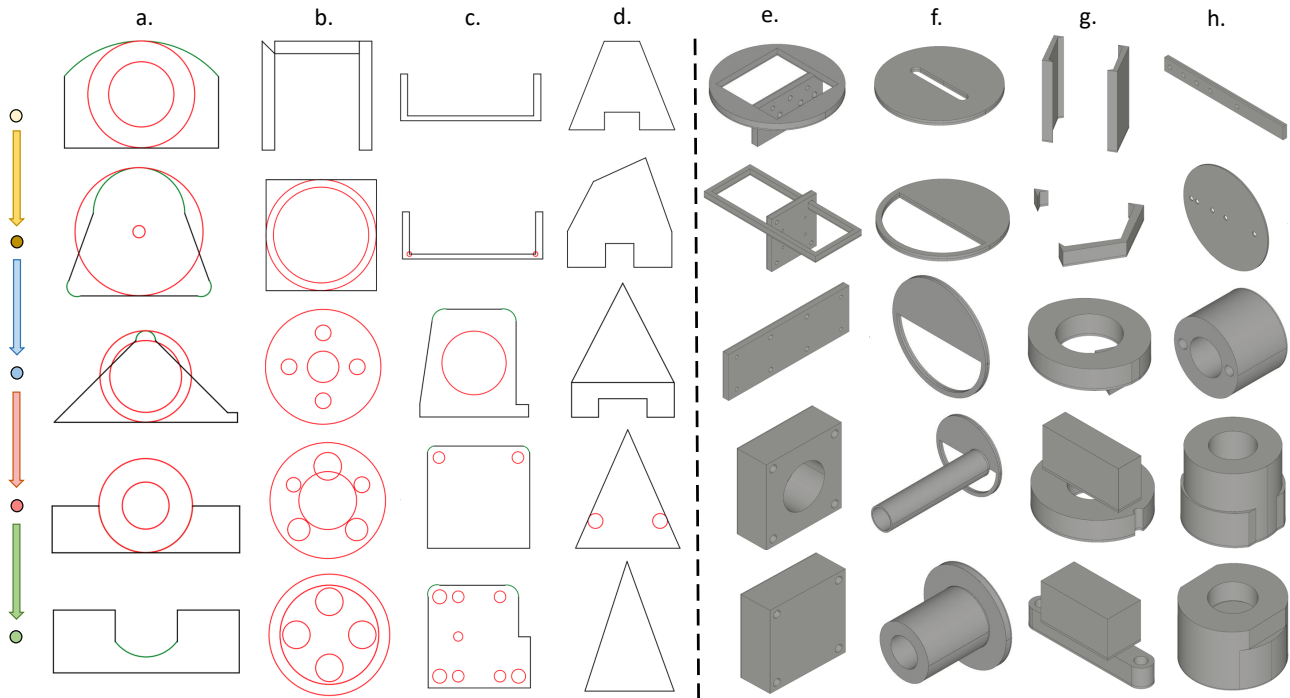
Acknowledgements

Figure 7. Design interpolation results for sketches (left) and CAD models (right). The top and the bottom rows show the source and target shapes, interpolated results appear in the middle rows.
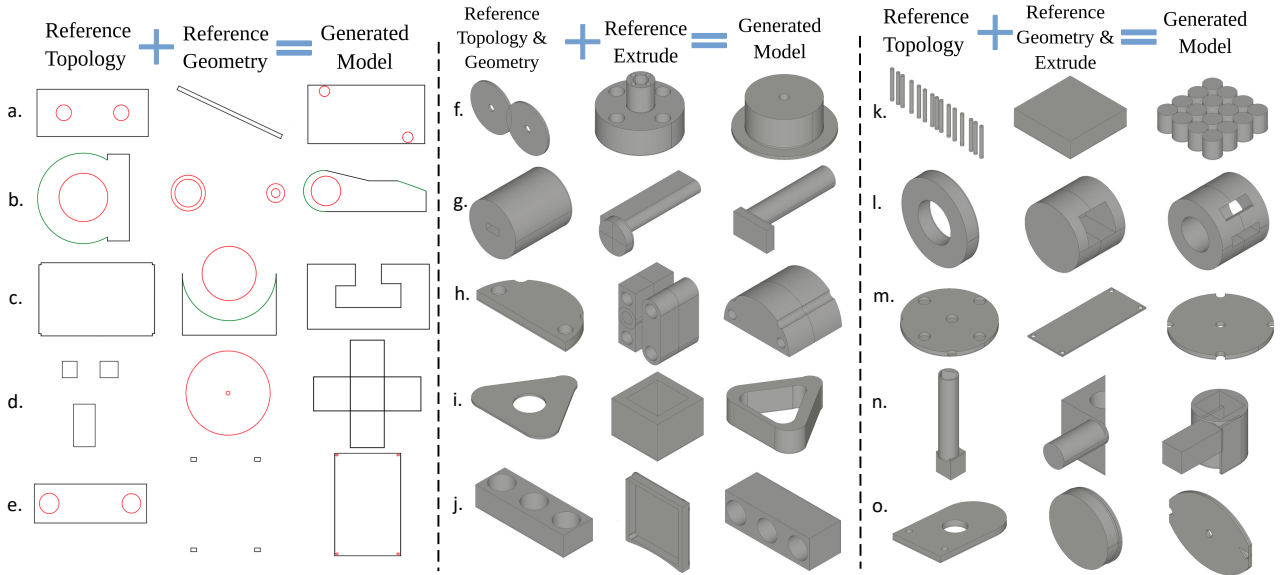

Figure 8. Topology, geometry, and extrude code mixing results. Given two reference CAD models or sketches (A and B), SkexGen can generate a new data whose topological property is similar to A and the geometric property is similar to B, by simply copying the topology codes from A, the geometry codes from B, and passing the mixed codes to the decoders for sampling.

References

Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pp. 40–49. PMLR, 2018.

Camba, J. D., Contero, M., and Company, P. Parametric cad modeling: An analysis of strategies for design reusability. *Computer-Aided Design*, 74:18–31, 2016.

Chakrabarti, A., Shea, K., Stone, R., Cagan, J., Campbell, M., Hernandez, N. V., and Wood, K. L. Computer-based design synthesis research: an overview. *Journal of Computing and Information Science in Engineering*, 11(2), 2011.

Chen, Z., Tagliasacchi, A., and Zhang, H. Bsp-net: Generating compact meshes via binary space partitioning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 45–54, 2020.

Das, A., Yang, Y., Hospedales, T., Xiang, T., and Song, Y.-Z. BézierSketch: A Generative Model for Scalable Vector Sketches. In *Computer Vision – ECCV 2020*, pp. 632–647. Springer International Publishing, 2020.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Du, T., Inala, J. P., Pu, Y., Spielberg, A., Schulz, A., Rus, D., Solar-Lezama, A., and Matusik, W. Inversecsg: Automatic conversion of 3d models to csg trees. *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 37(6):1–16, 2018.

Eitz, M., Hays, J., and Alexa, M. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):44:1–44:10, 2012.

Ellis, K., Nye, M., Pu, Y., Sosa, F., Tenenbaum, J., and Solar-Lezama, A. Write, execute, assess: Program synthesis with a repl. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9169–9178, 2019.

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12873–12883, 2021.

Ganin, Y., Bartunov, S., Li, Y., Keller, E., and Saliceti, S. Computer-aided design as language. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Grasel, J., Keskin, A., Swoboda, M., Przewozny, H., and Saxer, A. A full parametric model for turbomachinery blade design and optimisation. In *Proceedings of the ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 1: 30th Design Automation Conference, pp. 907–914, 09 2004.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M. M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2020.

Kania, K., Zięba, M., and Kajdanowicz, T. Ucsg-net–unsupervised discovering of constructive solid geometry tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Lee, S. H. and Lee, K. Partial entity structure: A compact non-manifold boundary representation based on partial topological entities. In *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*, SMA '01, pp. 159—170, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133669.

Maher, M. L., Poon, J., and Boulanger, S. Formalising design exploration as co-evolution. In *Advances in formal design methods for CAD*, pp. 3–30. Springer, 1996.

Nandi, C., Caspi, A., Grossman, D., and Tatlock, Z. Programming language tools and techniques for 3d printing. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

Nandi, C., Wilcox, J. R., Panchekha, P., Blau, T., Grossman, D., and Tatlock, Z. Functional programming for compiling and decompiling computer-aided design. *Proceedings*

of the ACM on Programming Languages, 2(ICFP):1–31, 2018.

Nash, C., Ganin, Y., Eslami, S. M. A., and Battaglia, P. W. Polygen: An autoregressive generative model of 3d meshes. *ICML*, 2020.

Para, W. R., Bhat, S. F., Guerrero, P., Kelly, T., Mitra, N., Guibas, L., and Wonka, P. Sketchgen: Generating constrained cad sketches. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

Razavi, A., van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, pp. 14866–14876, 2019.

Seff, A., Ovadia, Y., Zhou, W., and Adams, R. P. Sketch-Graphs: A large-scale dataset for modeling relational geometry in computer-aided design. In *ICML 2020 Workshop on Object-Oriented Learning*, 2020.

Seff, A., Zhou, W., Richardson, N., and Adams, R. P. Vitruvion: A generative model of parametric cad sketches. In *International Conference on Learning Representations (ICLR)*, 2022.

Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., and Maji, S. Csgnet: Neural shape parser for constructive solid geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Tian, Y., Luo, A., Sun, X., Ellis, K., Freeman, W. T., Tenenbaum, J. B., and Wu, J. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations (ICLR)*, 2019.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6309—6318, Red Hook, NY, USA, 2017. Curran Associates Inc.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5998–6008, 2017.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Weiler, K. J. *Topological structures for geometric modeling*. Rensselaer Polytechnic Institute, 1986.

Willis, K. D. D., Jayaraman, P. K., Lambourne, J. G., Chu, H., and Pu, Y. Engineering sketch generation for computer-aided design. In *The 1st Workshop on Sketch-Oriented Deep Learning (SketchDL), CVPR 2021*, June 2021a.

Willis, K. D. D., Pu, Y., Luo, J., Chu, H., Du, T., Lambourne, J. G., Solar-Lezama, A., and Matusik, W. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4), 2021b.

Wu, R., Xiao, C., and Zheng, C. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021.

Xu, X., Peng, W., Cheng, C.-Y., Willis, K. D., and Ritchie, D. Inferring cad modeling sequences using zone graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6062–6070, June 2021.

Yu, F., Chen, Z., Li, M., Sanghi, A., Shayani, H., Mahdavi-Amiri, A., and Zhang, H. Capri-net: Learning compact cad shapes with adaptive primitive assembly. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11768–11778, 2022.

A. Construction Sequence: Full Specification

A construction sequence must follow certain rules to be a valid sketch-and-extrude model. Instead of providing a full specification as a context free grammar, which is possible, we describe rules with a few bullet points.

- A sketch-and-extrude model consists of multiple extruded-sketches.

- A sketch consists of multiple faces.

- A face consists of multiple loops.

- A loop consists of multiple curves.

- A curve is either a line, an arc, or a circle, which consist of 2, 3, or 4 points, respectively.

- A loop with a circle can not contain additional curves since it is already a closed path.

- A point is represented by a geometry token.

- An end-primitive token appears at the end of each primitive (curve, line, face, loop, sketch, or extruded-sketch).

- When a face consists of multiple loops, the first loop defines the external boundary, and the remaining loops define internal loops (i.e., holes).

B. Extrude Branch Architecture

The extrude branch takes a subsequence of the input, where a token is either 1) An extrude token, which is one of the five types (H,R,O, S, B); 2) An end-primitive token (E) for the extruded-sketch primitive type; and 3) An end-sequence token (End).

- H indicates the displacements of the top and the bottom planes from the reference plane in which a sketch is extruded to form a solid. Since we know that the two values are necessary, we repeat H tokens twice in the sequence instead of having one token encode two values. We quantize each height value into 64 bins and represent the value as a 64 dimensional one-hot vector.

- R is a 3D rotation of the extrusion direction. Looking through 3×3 rotation matrices in our datasets, each entry of the rotation matrix has a value of either -1, 0, or 1. These three values cover almost all rotations from DeepCAD dataset (>99%). Similarly to H, we repeat R nine times in the sequence, where each token indicates one of the three values as a 3 dimensional one-hot vector.

- O is a 3D translation vector applied to the extruded solid. Similarly to H, we know that three values are necessary, and we repeat O three times in the sequence. We quantize each height value into 64 bins and represent the value as a 64 dimensional one-hot vector.

- S indicates a sketch scaling factor consisting of three values: the center of scaling as a 2D coordinate and the uniform scaling factor. Similarly, we quantize each value into 64 bins as a 64 dimensional one-hot vector.

- B indicates one of the three Boolean operations (intersection, union, or subtraction) and is represented by a 3 dimensional one-hot vector.

Embeddings: Figure 1 in the main paper shows that an extrude subsequence consists of 7 types of tokens including End. In fact, an extrude subsequence after flatten consists of $19(= 2 + 9 + 3 + 3 + 1 + 1)$ individual tokens: $[HHRRRRRRRRROOOSSSBE]$ (not counting End). Each token is represented by a $72(= 64 + 3 + 3 + 1 + 1)$ dimensional one-hot vector, where numeric tokens (H, O, S) share the 64 bins.

Extrude encoder: The encoder is the same as the topology or geometry encoder except that a one-hot vector is of 72 dimensional instead of 4101. Similar to Eq. 5, we also add an additional learnable embedding which distinguish the token types (e.g. the 9 rotation matrix tokens VS the 3 translation tokens).

Extrude decoder: The decoder is the same as the sketch decoder except 1) Only one (instead of two) codebook is used and no learnable embedding is added to distinguish different codebooks in cross-attention (Eq. 5); 2) The output h_K^{out} is of dimension 72 instead of 4101.

Training: Similar to Eq. 6, reconstruction loss, codebook loss, and commitment loss are used for training. Extrude branch has a total of four code-tokens and a codebook size of 1000.

C. Code Selector: Details

The topology, geometry, and extrude codebooks provide $10(= 4 + 2 + 4)$ codes to the sketch and extrude decoders for model generation. Without loss of generality, let us explain a topology-geometry conditioned extrude code selector. In our experiments, the maximum size of each codebook is 1,000, and we simply use a 1000 dimensional one-hot encoding to represent each code in a codebook.

Just as in the sketch decoder (Sect. 4.3), one-hot vectors are multiplied with a learnable matrix of size (256×1000) to compress feature embedding to 256 dimensional. Codes from the topology and the geometry codebooks are then injected via cross-attention to the decoder. We use the same position encoding to the codebook vectors as in Eq. 5. The decoder architecture is similar to the sketch decoder, producing the missing four extrude codes one by one in an autoregressive manner. The output embeddings finally pass through a fully-connected layer (1000×256) and softmax, predicting the probability likelihood over the 1000 codebook indexes.

D. Additional Results

D.1. Sketch Generation

Figure 9 to Figure 15 show uniformly sampled random sketches generated by SkexGen.

D.2. CAD Generation

Figure 16 to Figure 22 show uniformly sampled random CAD models generated by SkexGen.

Figure 9. Uniformly sampled random sketches generated by SkexGen.

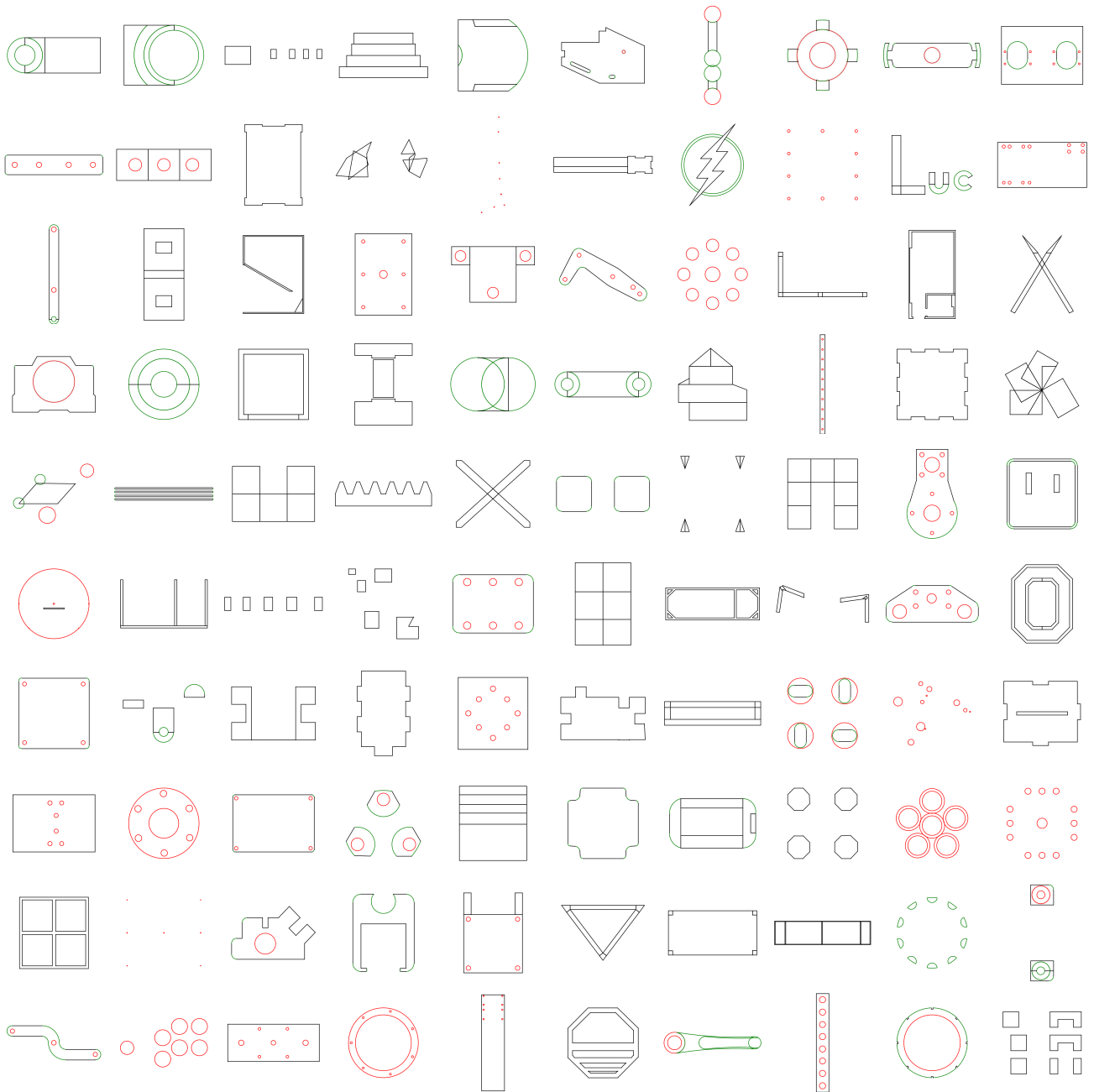*Figure 10.* Uniformly sampled random sketches generated by SkexGen.

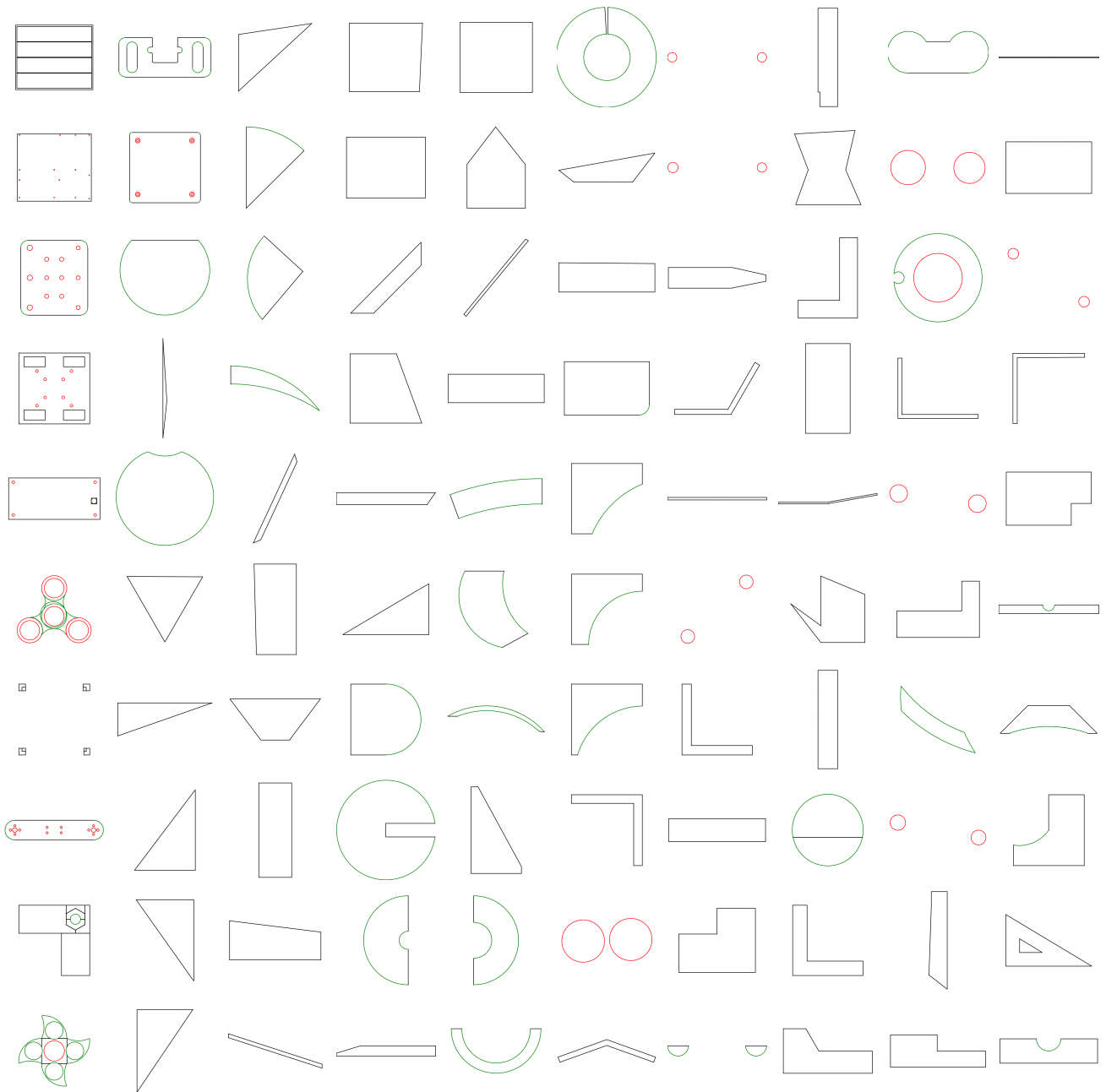*Figure 11.* Uniformly sampled random sketches generated by SkexGen.

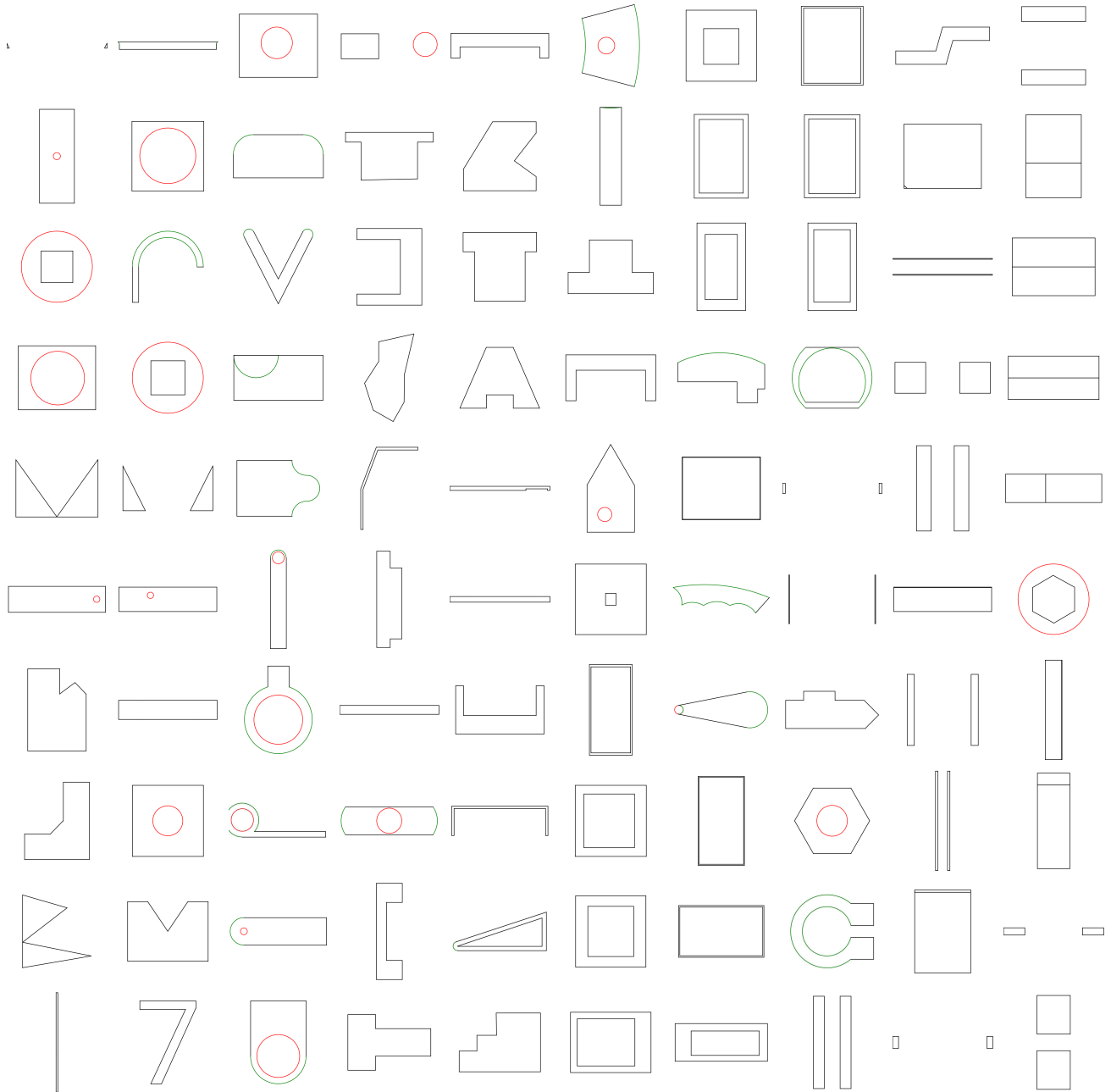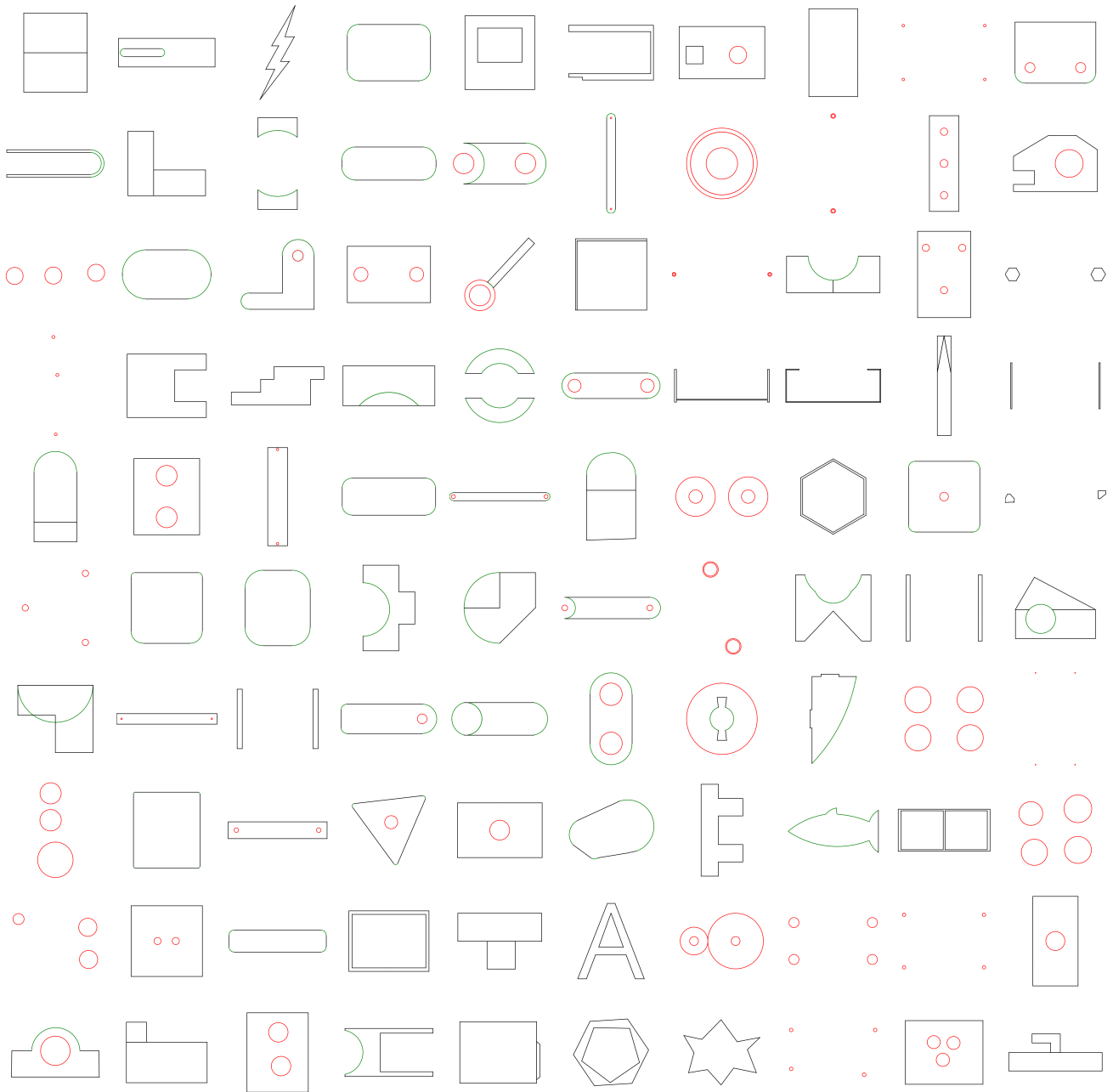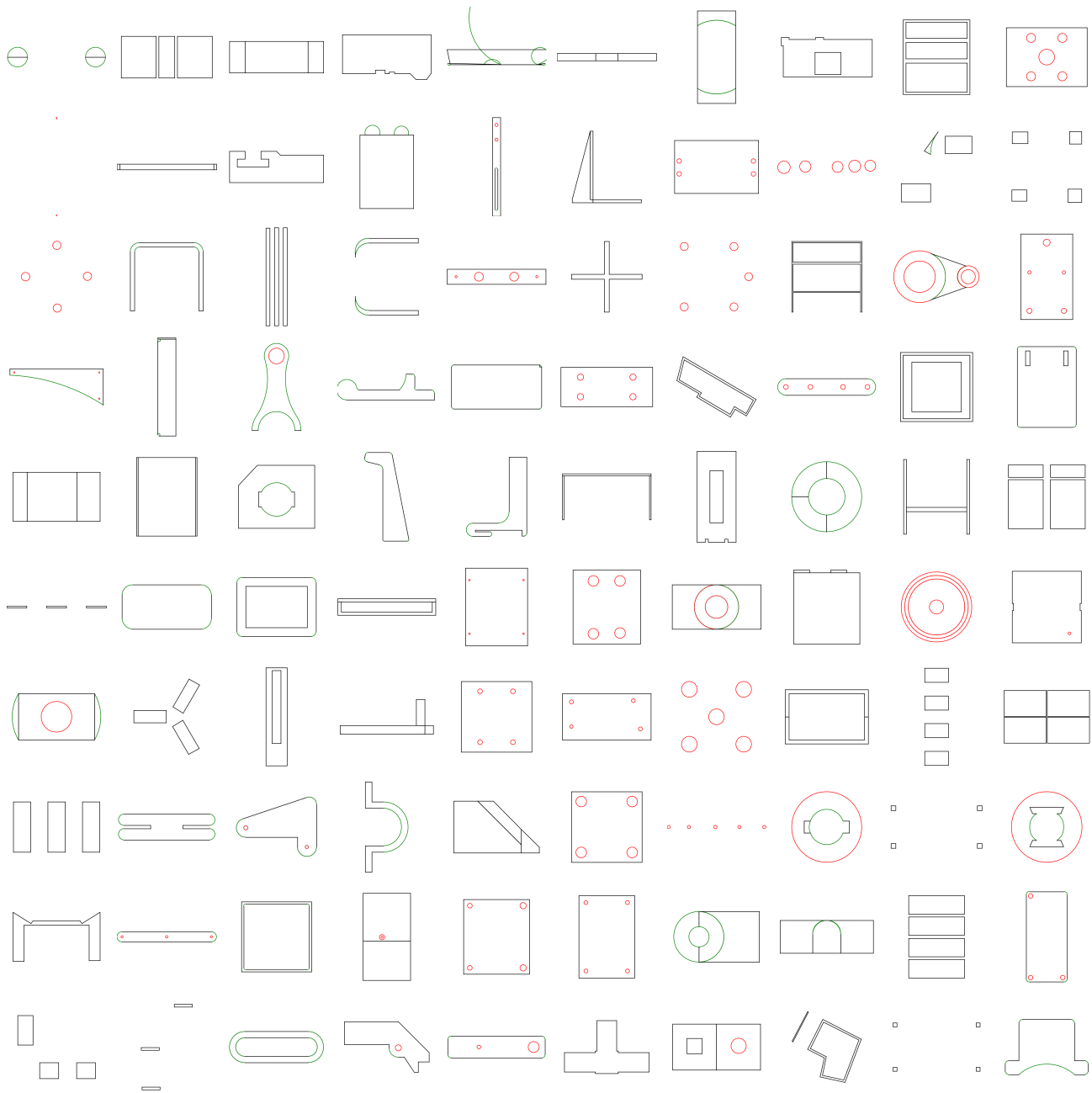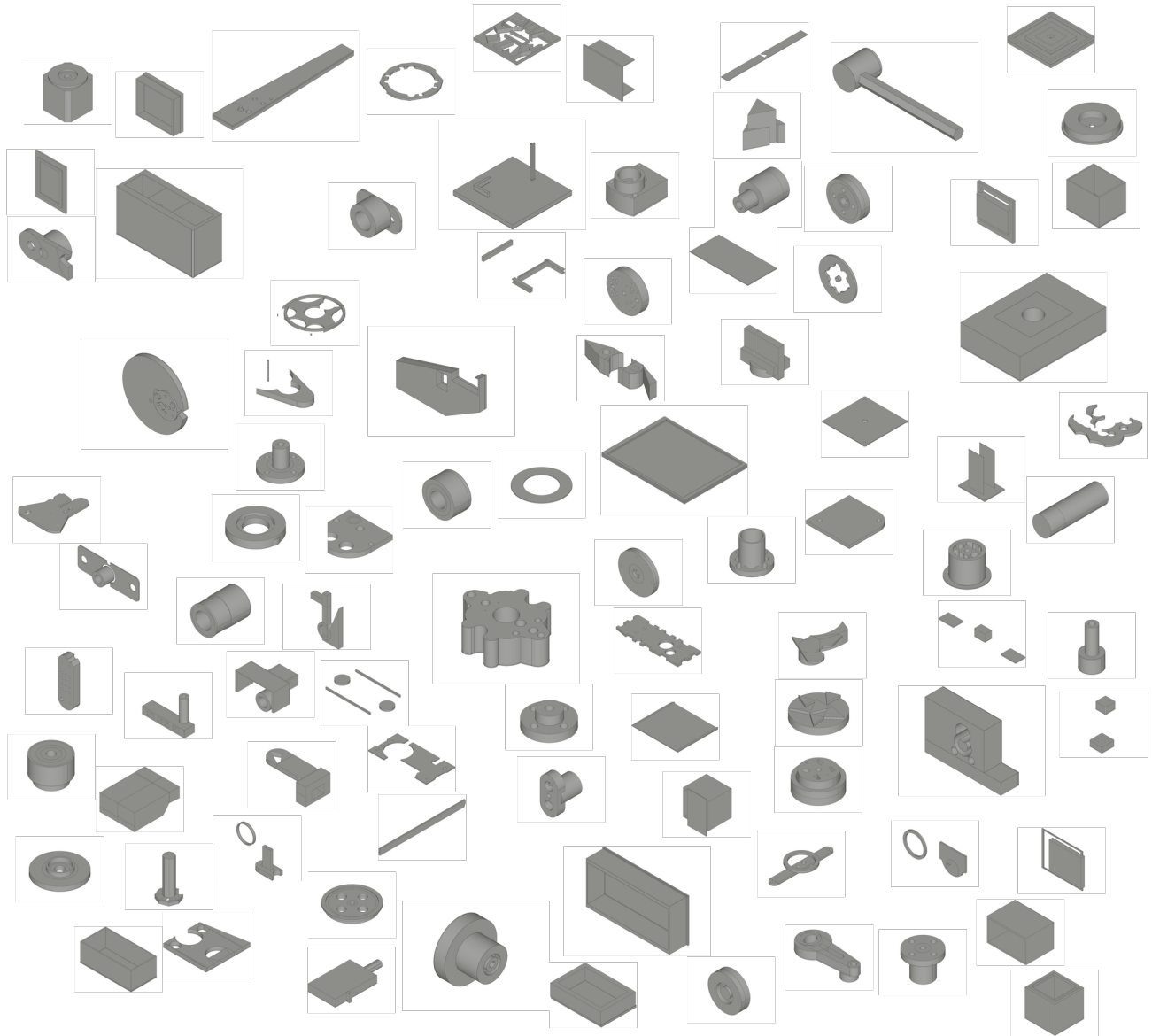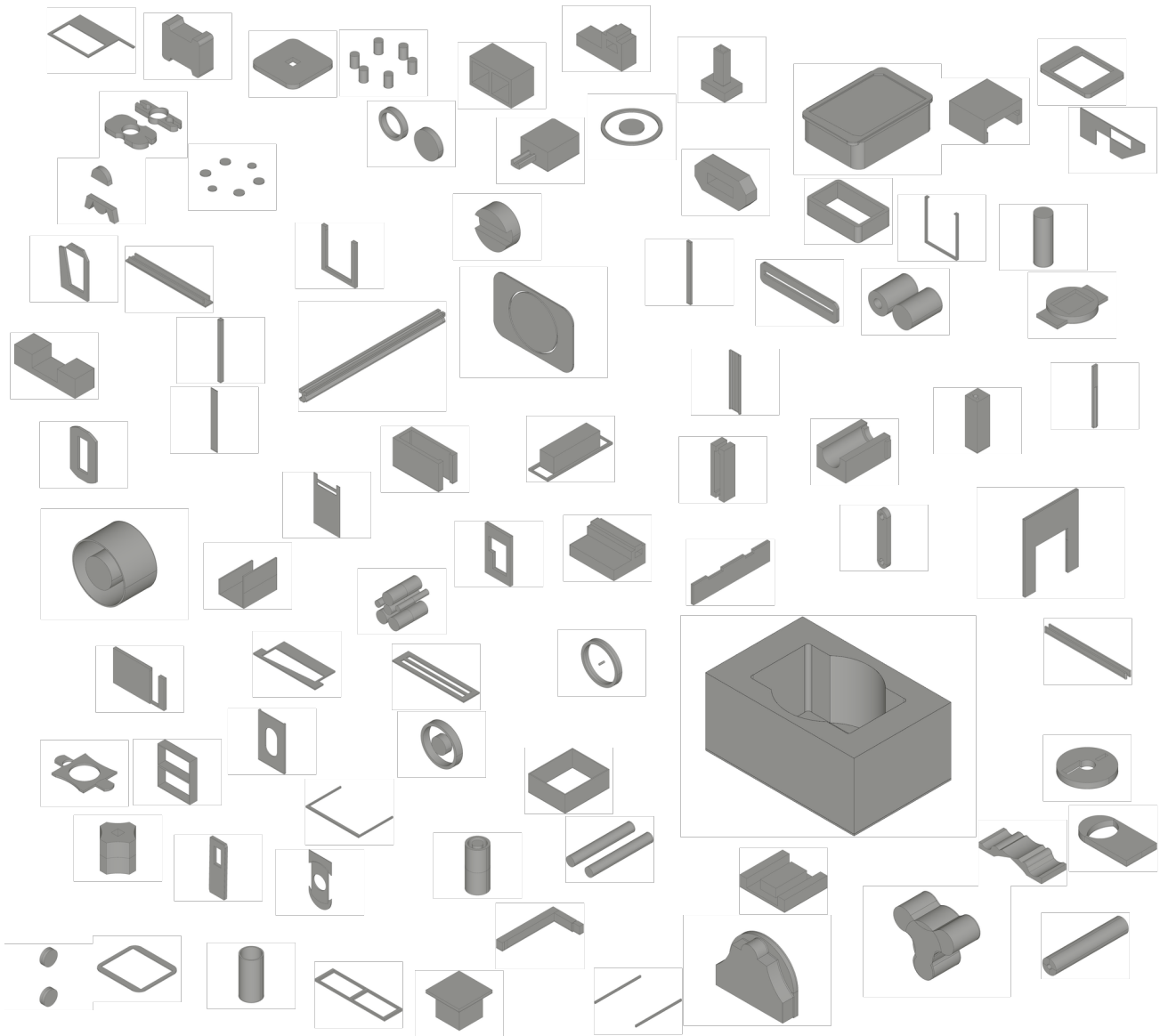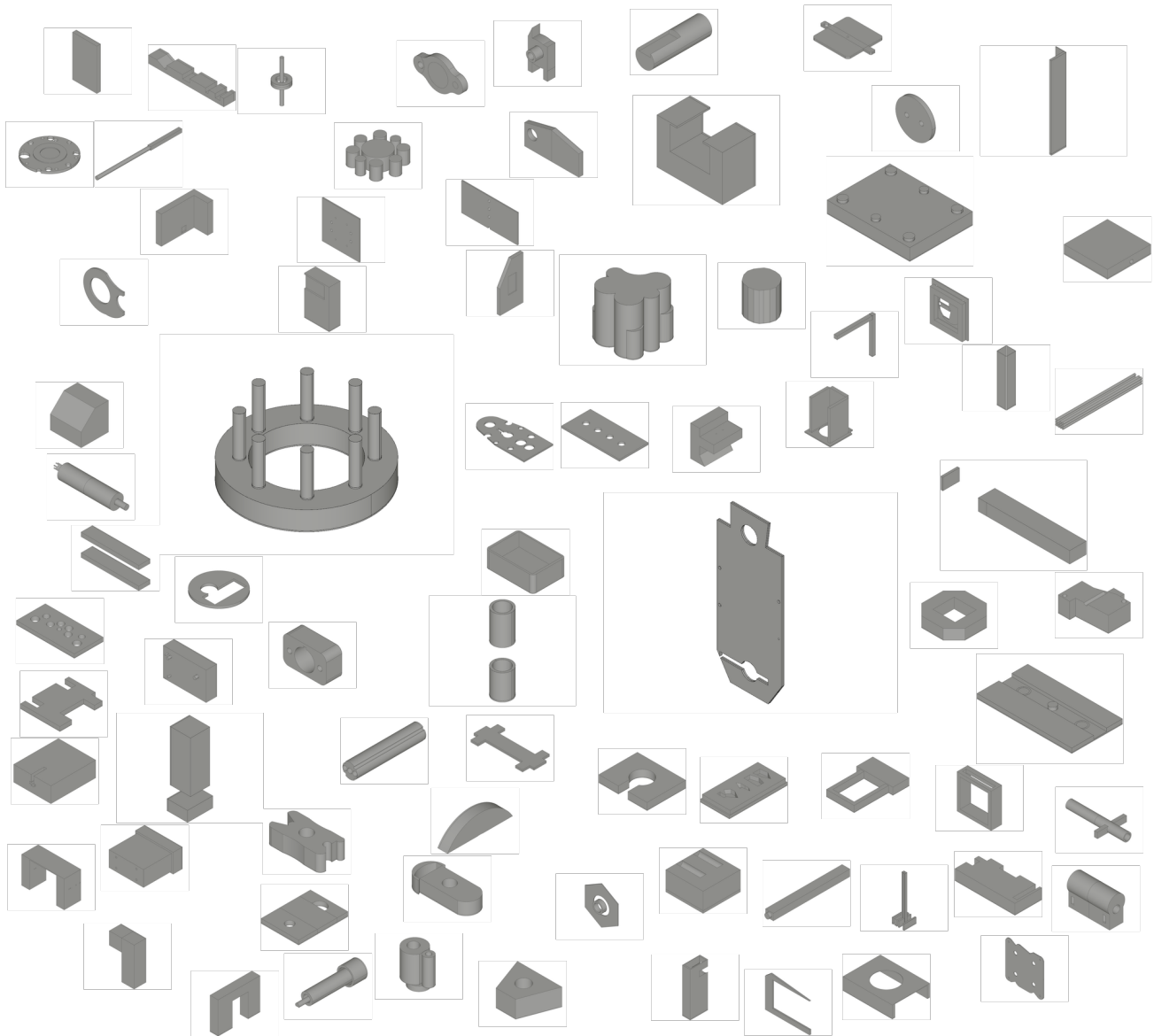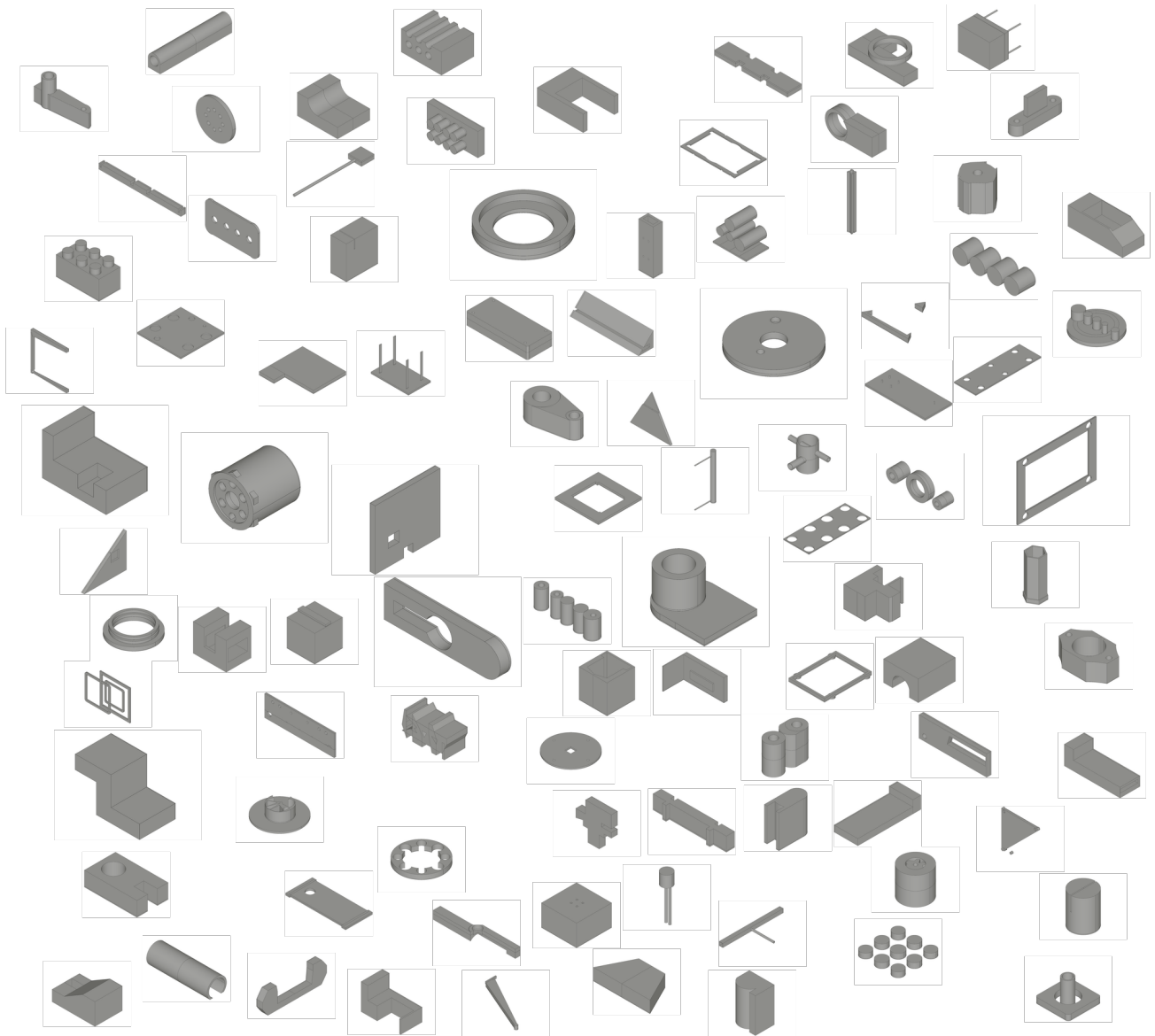*Figure 12.* Uniformly sampled random sketches generated by SkexGen.

Figure 13. Uniformly sampled random sketches generated by SkexGen.

Figure 14. Uniformly sampled random sketches generated by SkexGen.

Figure 15. Uniformly sampled random sketches generated by SkexGen.

Figure 16. Uniformly sampled random CAD models generated by SkexGen.

Figure 17. Uniformly sampled random CAD models generated by SkexGen.

Figure 18. Uniformly sampled random CAD models generated by SkexGen.

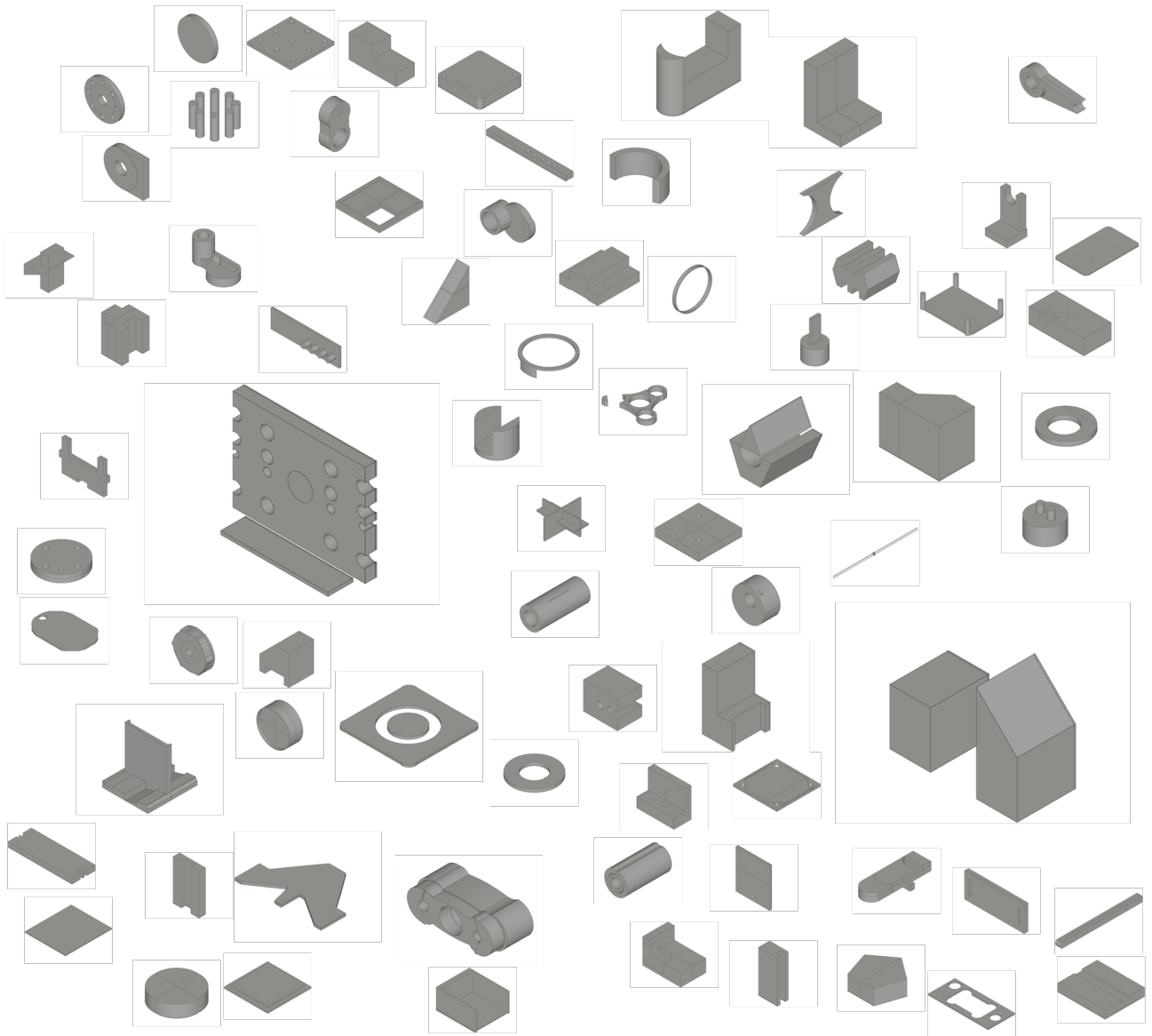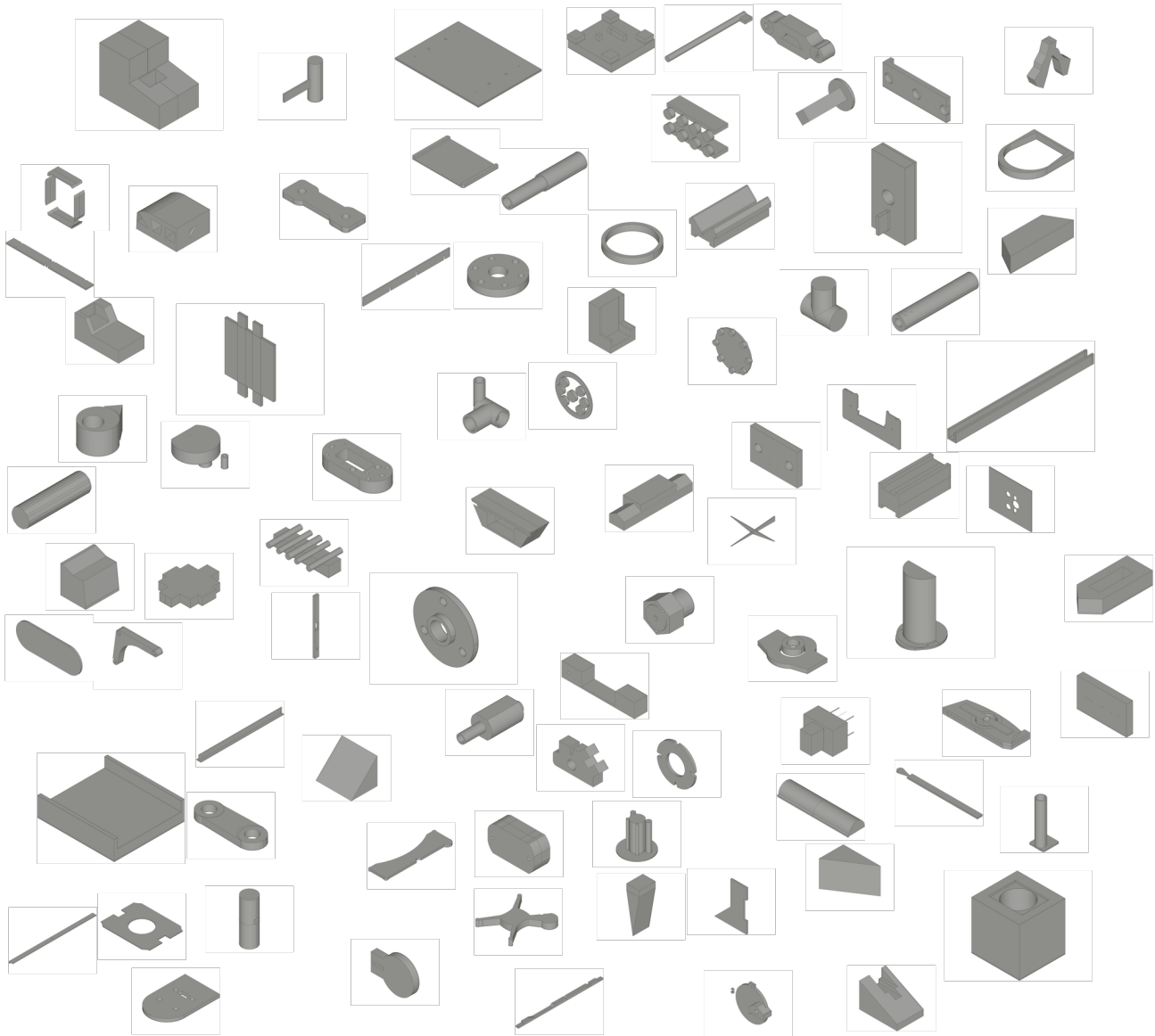*Figure 19.* Uniformly sampled random CAD models generated by SkexGen.

Figure 20. Uniformly sampled random CAD models generated by SkexGen.

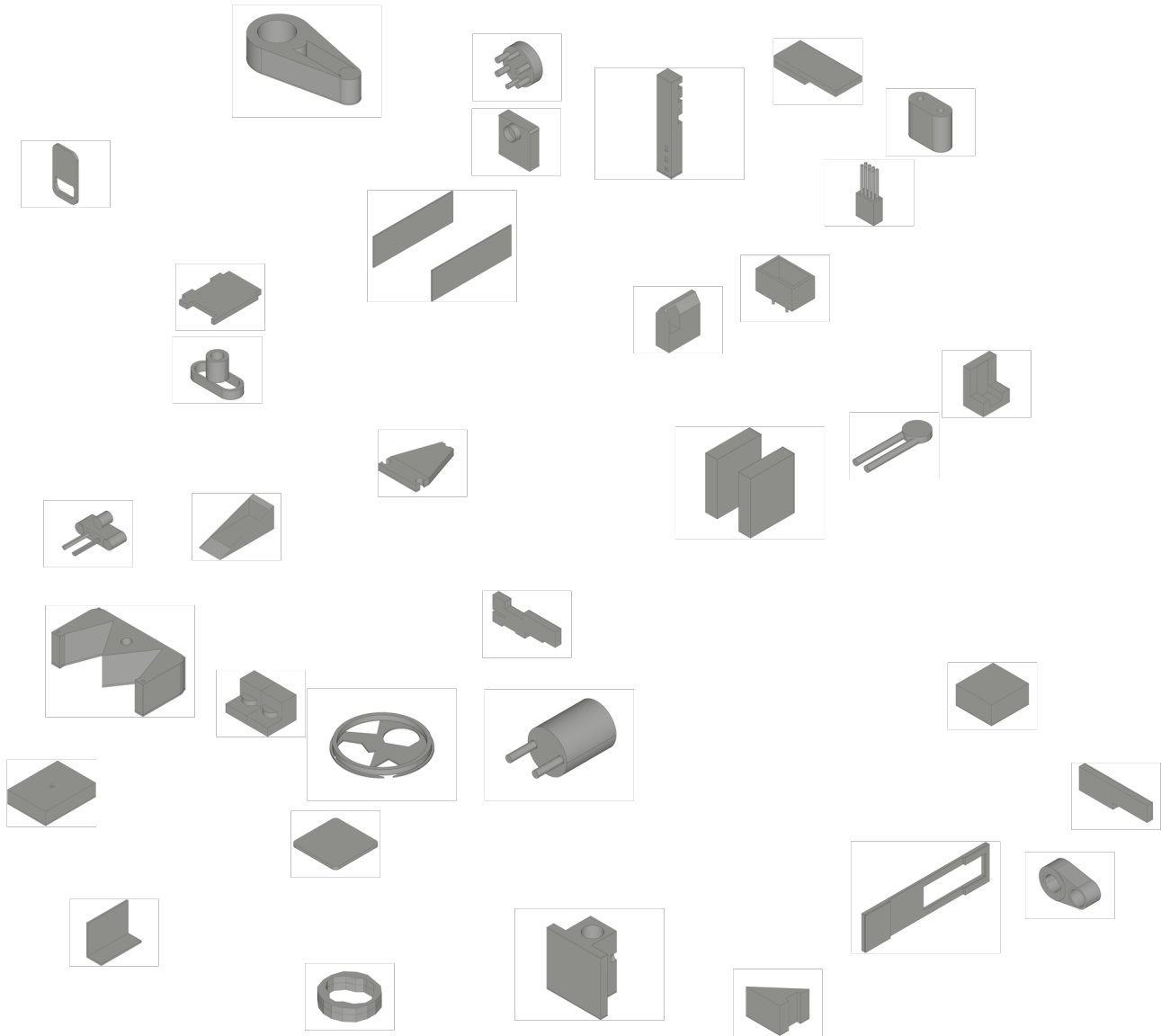*Figure 21.* Uniformly sampled random CAD models generated by SkexGen.

Figure 22. Uniformly sampled random CAD models generated by SkexGen.