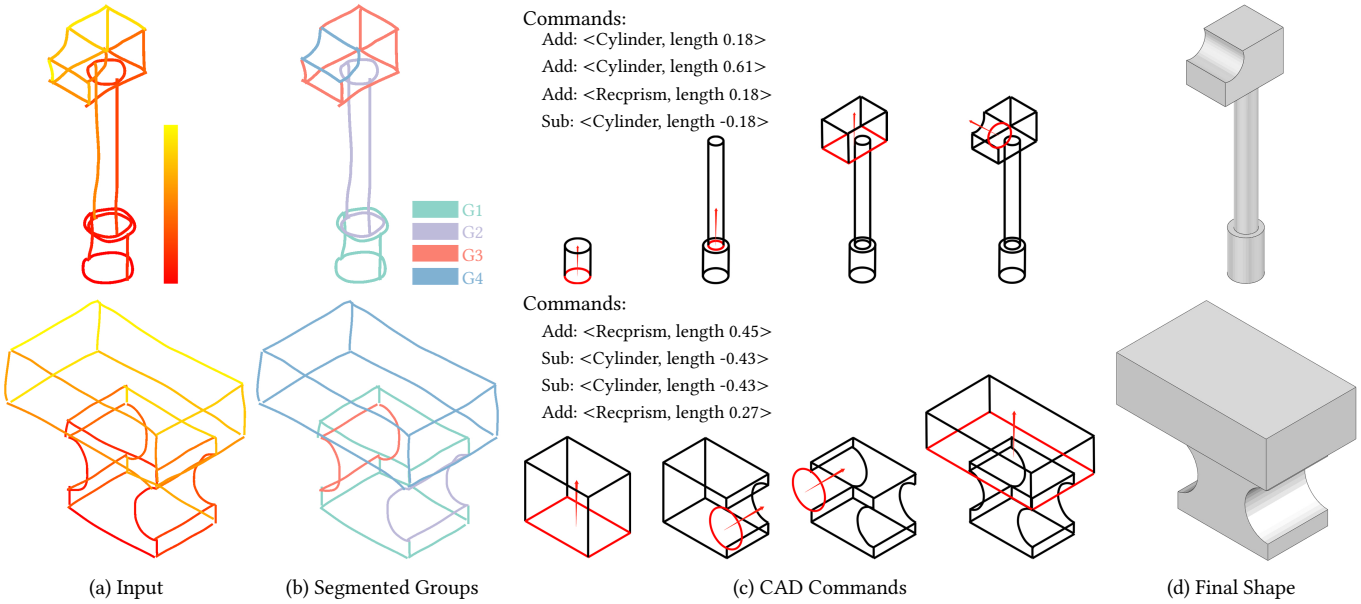
Free2CAD: Parsing Freehand Drawings into CAD Commands

Changjian Li, Hao Pan, Adrien Bousseau, Niloy J. Mitra

Free2CAD: Parsing Freehand Drawings into CAD Commands

CHANGJIAN LI, Inria, Université Côte d'Azur, France and University College London, United Kingdom
HAO PAN, Microsoft Research Asia, China
ADRIEN BOUSSEAU, Inria, Université Côte d'Azur, France
NILOY J. MITRA, University College London, United Kingdom and Adobe Research, United Kingdom



(a) Input (b) Segmented Groups (c) CAD Commands (d) Final Shape

Fig. 1. **Parsing freehand drawings into CAD programs.** In this example, the user simply draws the final shape of the hammer and the anvil (a, strokes colored by drawing order as red → yellow). From this input, Free2CAD produces a corresponding set of CAD commands (c), thus freeing the user from having to mentally parse how to construct the shape with valid CAD operations. At the core of our system is a deep sequence-to-sequence neural network that automatically segments the input drawing into groups of strokes (b), each group representing a part of the shape that can be produced by a unique CAD command. Our system sequentially processes each groups to recognize and fit the corresponding CAD commands, yielding a CAD program which, when executed, produces a 3D shape that is faithful to the input drawing (d).

CAD modeling, despite being the industry-standard, remains restricted to usage by skilled practitioners due to two key barriers. First, the user must be able to mentally parse a final shape into a valid sequence of supported CAD commands; and second, the user must be sufficiently conversant with CAD software packages to be able to execute the corresponding CAD commands. As a step towards addressing both these challenges, we present Free2CAD wherein the user can simply sketch the final shape and our system parses the input strokes into a sequence of commands expressed in a simplified CAD language. When executed, these commands reproduce the sketched object. Technically, we cast sketch-based CAD modeling as a sequence-to-sequence translation problem, for which we leverage the powerful Transformers neural network architecture. Given the sequence of pen strokes as input, we introduce the new task of grouping strokes that correspond to individual CAD operations. We combine stroke grouping with geometric fitting of the operation parameters, such that intermediate groups are geometrically corrected before being reused, as context, for subsequent steps in the sequence inference. Although trained on synthetically-generated data, we demonstrate that Free2CAD generalizes to sketches created from real-world CAD models as well as to sketches drawn by novice users.

Code and data are at https://github.com/Enigma-li/Free2CAD.

CCS Concepts: • **Computing methodologies → Shape modeling**.

Additional Key Words and Phrases: sketch, CAD modeling, procedural modeling, Transformer

ACM Reference Format:
Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J. Mitra. 2022. Free2CAD: Parsing Freehand Drawings into CAD Commands. *ACM Trans. Graph.* 41, 4, Article 93 (July 2022), 16 pages. https://doi.org/10.1145/3528223.3530133

Authors' addresses: Changjian Li, Inria, Université Côte d'Azur, 2004 route des lucioles, Valbonne, France and University College London, United Kingdom, chjili2011@gmail.com; Hao Pan, Microsoft Research Asia, No.5 Danling Rd, Beijing, China, haopan@microsoft.com; Adrien Bousseau, Inria, Université Côte d'Azur, 2004 route des lucioles, Valbonne, France, adrien.bousseau@inria.fr; Niloy J. Mitra, University College London, 169 Euston Square, London, United Kingdom and Adobe Research, United Kingdom, n.mitra@cs.ucl.ac.uk.

1 INTRODUCTION

Computer-Aided-Design (CAD) is the industry standard for creating 3D shapes, but its inherent complexity has so far restricted its usage to a small elite group of expert users. CAD teachers have long identified two major challenges that prevent effective use of CAD software by novices [Bhavnani et al. 1999; Chester 2006]. First, users need to mentally decompose the shape they wish to model into a succession of geometric operations supported by CAD software – a skill referred to as *strategic knowledge*. Second, users need to find how to execute each operation within the software interface – a skill referred to as *command knowledge*. In this paper, we make a step towards addressing these two challenges by proposing Free2CAD, a sketch-based modeling system that turns a complete drawing of a 3D shape into a sequence of CAD operations reproducing the sketched object. We demonstrate the feasibility of such as system by focusing on a simplified CAD language of positive and negative extrusions, which is sufficiently expressive to create CAD models of moderate complexity from novice drawings as illustrated in Figures 1.

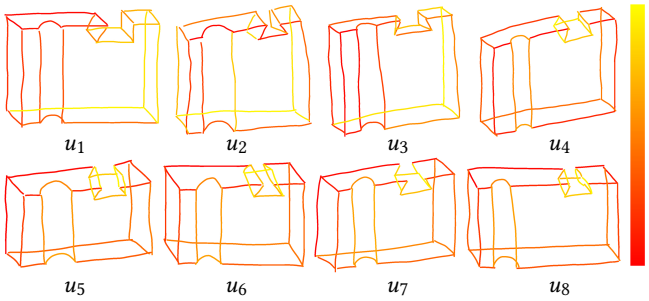


Fig. 2. **Drawing order.** Pilot study to assess variations in drawing order across eight different users when asked to draw a CAD model. We use the classical autumn color coding (i.e., red → yellow) to visualize the stroke drawing order. Users, with varying expertise, demonstrate different drawing patterns. Less skilled users draw from left to right (u_1 and u_3) or front to back (u_2), while more skilled users (u_4 to u_8) draw the object part-by-part.

In order to assess drawing strategies of novice users, we first conducted a pilot study (see Figure 2). When asked to draw a target 3D shape, different users adopted different strategies. Some users (u_4 to u_8) drew the shape as a sequence of simple primitives – starting with a box and then subtracting a cylinder and a prism – as is expected by prior interactive systems for constructing CAD models with sketching gestures [Jorge et al. 2003; Zeleznik et al. 1996] or step-by-step sketching [Li et al. 2020; Nishida et al. 2016]. However, other users (u_1 to u_3) drew directly the outline of the entire shape and its details, from left to right, top to bottom, starting to draw new parts before completing the other parts. We designed Free2CAD to be robust to these different drawing permutations, effectively freeing users from the burden of strategically constructing their drawings as sequences of clearly separated steps.

Our system takes as input a single perspective drawing of the envisioned 3D shape, represented as a sequence of pen strokes drawn within our user interface in a known camera view. Inspired by the cognitive process of expert CAD modelers, our system groups the pen strokes into parts that can be produced by individual CAD operations, and recognizes which operation should be executed for each part. Since ordering the operations is an ambiguous task, we assume that users draw the shape from coarse-to-fine, such that the first stroke of each group appears before the first strokes of subsequent groups. Yet, we allow the groups of strokes to overlap in time to accommodate the different drawing orders observed in our pilot study.

Grouping pen strokes into CAD operations raises novel challenges compared to traditional sketch segmentation [Yang et al. 2021]. First, in contrast to semantic segmentation where each class has a fixed number of labels, the drawings we need to segment are composed of an arbitrary number of groups. Second, the CAD operation depicted by a group of strokes should apply to the (contextual) geometry produced by preceding operations, which introduces an implicit sequential dependence between the groups. To tackle these challenges, we propose a tailored sequence-to-sequence neural network architecture capable of processing variable numbers of groups, and of segmenting one group of strokes at a time, conditioned on the groups that have been already segmented. Importantly, our algorithm interleaves stroke grouping, CAD operation recognition, and operation execution, such that each of these steps benefits from *contextual feedback* on what has been constructed so far.

In the absence of a dataset of aligned, real-world drawing and CAD modeling sequences, we train our system by generating a large amount of synthetic CAD sequences that we render in the style of a line drawing. This synthetic data provides us with ground truth group labels to train our stroke grouping module. However, despite reaching reasonable grouping accuracy, this module inevitably introduces prediction errors at test time, which propagate to subsequent groups when the model is executed in an auto-regressive fashion. Various strategies have been proposed in the literature on sequential architectures to deal with this so-called *exposure bias* [Mihaylova and Martins 2019], such as mixing ground truth labels with predicted labels during training to teach the model to recover from mistakes. We use the unique geometric setting of our problem to propose a novel solution to this challenge. Specifically, we leverage the fitting procedure to identify strokes that have been mislabeled within a group, and use this information to correct the grouping information *before* executing the model to predict the next group. By interleaving grouping prediction and geometric fitting, our design effectively injects 3D information within the grouping task, achieving higher accuracy than its naive counterpart.

We evaluate our system on user-drawn sketches, as well as on drawings generated from publicly available CAD sequences [Willis et al. 2021]. In both cases, Free2CAD can successfully convert a drawing to a sequence of ordered CAD operations (see Figure 1). Note that in both cases we test our system on out-of-training data, thereby evaluating the generalization behavior of Free2CAD.

In summary, we introduce the following contributions.

- A novel neural network architecture for stroke grouping. This sequential architecture supports an arbitrary number of strokes and an arbitrary number of groups, sequentially producing one group at a time.
- An interleaved segmentation and geometric fitting scheme that leverages inferred 3D geometry to improve (subsequent) grouping of strokes that represent CAD operations.

- Combined together, these technical contributions enable the first sketch-based modeling system where users can draw entire 3D shapes freehand without knowing how they should be decomposed into CAD operations.

We report extensive evaluation on ∼ 300 sketches automatically produced using the Fusion360 dataset [2021]. Our user study shows that even users without CAD modeling background can produce usable CAD sequences with our system, and that our method allows different users to approach this drawing task with different strategies.

2 RELATED WORK

Our work bridges sketch-based modeling and CAD modeling. The key challenge is to segment the input drawing into parts that correspond to CAD operations. We next discuss related work in sketch-based modeling, CAD modeling, and sketch segmentation.

Sketch-based modeling. The recent survey by Bonnici et al. [2019] provides a detailed discussion of existing sketch-based modeling systems, and points to the open challenges of the field. In particular, they identify the need for methods that would integrate seamlessly in existing design workflows, which motivates our system to translate freehand drawings into CAD commands.

In the last two decades, a number of algorithms and user interfaces have been proposed to lift 2D drawings to 3D [Bae et al. 2008; Gryaditskaya et al. 2020; Li et al. 2017; Lipson and Shpitalni 1996; Orbay and Kara 2012; Schmidt et al. 2009; Xu et al. 2014]. Most of these algorithms cast the problem of 3D reconstruction as an optimization where the depths of the pen stroke are treated as variables, and geometric relationships between the strokes – such as parallelism and orthogonality – act as constraints. Unfortunately, the 3D curves and surfaces that these methods produce have many degrees of freedom, which make them difficult to edit, and prone to over-fitting to drawing inaccuracy.

Higher robustness and editability is achieved by methods that convert drawings into 3D shapes defined by a small number of parameters. Early work relied on specific drawing gestures or user annotations to instantiate parametric primitives and operations [Gingold et al. 2009; Jorge et al. 2003; Shao et al. 2013; Shtof et al. 2013; Zeleznik et al. 1996]. Recent methods leverage deep learning to recognize pre-defined shapes and to regress their parameters [Huang et al. 2016; Li et al. 2020; Nishida et al. 2016]. However, these methods assume an iterative workflow where users draw a *single* operation at a time, which is immediately translated into a CAD operation to serve as context for the next operation. As a result, while these systems effectively free users from having to execute CAD commands in feature-rich user interfaces, the users are still required to strategically decompose the target shape into a series of drawing steps corresponding to individual CAD commands. The same limitation applies to commercial tools which, although offering simple user interfaces for novices [Autodesk 2019; Shapr3D 2016; Trimble 2019], require users to plan the sequence of operations needed to reach their goal.

CAD modeling. Our approach contributes to the ongoing effort to automate part of the CAD modeling workflow. Treating CAD instructions as a domain-specific programming language, several approaches attempt to synthesize CAD programs to reverse-engineer 2D or 3D shapes [Du et al. 2018; Ellis et al. 2018; Jones et al. 2020; Kania et al. 2020; Nandi et al. 2018; Sharma et al. 2018; Tian et al. 2019; Wu et al. 2018, 2021; Yu et al. 2021]. A major challenge faced by this family of methods is to explore the combinatorial space of all possible CAD sequences that can reproduce a target shape. Machine learning emerged as a key ingredient to make this problem tractable, as it can be used to focus the search on the most promising sequences [Ellis et al. 2019; Willis et al. 2021; Xu et al. 2021]. Working with drawing sequences reduces the size of this search space, as we can exploit the fact that people often draw shapes from coarse-to-fine, even though the precise ordering of the individual strokes differs among users (cf., our pilot study reported in Figure 2). On the other hand, drawings lack 3D information, which prevents us from relying on 3D primitive fitting or space partitioning to identify candidate CAD operations, as commonly done to reduce the complexity of the problem [Du et al. 2018; Wu et al. 2018; Xu et al. 2021]. We instead leverage deep learning to identify strokes that belong to the same operation.

In addition to providing a compact representation of 3D shapes, CAD programs also support constraints between the elements that compose a shape, which helps preserve the original design intent during editing. Several recent methods focus on identifying such constraints, often in the context of so-called *CAD sketches* that represent intermediate 2D shapes that are extruded to form 3D surfaces [Ganin et al. 2021; Para et al. 2021; Seff et al. 2021]. In an interactive setting, Hempel et al. [2019] propose novel workflows to create graphical programs with constraints through direct manipulation of the resulting shapes. We restrict ourselves to a set of simple heuristic steps to identify constraints and regularize our 3D shapes as a post-process, and instead focus on the challenge of recognizing CAD command sequences from approximate drawings.

Sketch segmentation. The problem of grouping strokes into meaningful entities has been explored in the context of semantic sketch segmentation [Li et al. 2018, 2019; Schneider and Tuytelaars 2016; Yang et al. 2021]. However, our goal differs as we aim at identifying parts of a shape that can be produced by CAD operations, such as holes and protrusions, rather than recognizing parts that have a semantic meaning, such as the legs of a chair or the wings of a bird. Furthermore, we target perspective drawings of 3D shapes rather than abstract drawings of pre-defined object classes. To tackle this task, we propose a tailored neural network architecture that supports an arbitrary number of groups, and that benefits from fitting of geometric operations to improve its predictions. Our evaluation shows that our novel architecture outperforms state-of-the-art sketch segmentation methods on our task.

We note that our goal relates to an early effort by Chen et al. [2007] to decompose line drawings of complex 3D shapes into simpler parts, but their geometric algorithm requires that the input drawing forms a well-connected graph to detect so-called internal faces along which they cut the shape. In contrast, our deep learning approach is robust to drawing inaccuracies and it decomposes the drawing into parts based on whether they can be explained by CAD commands.

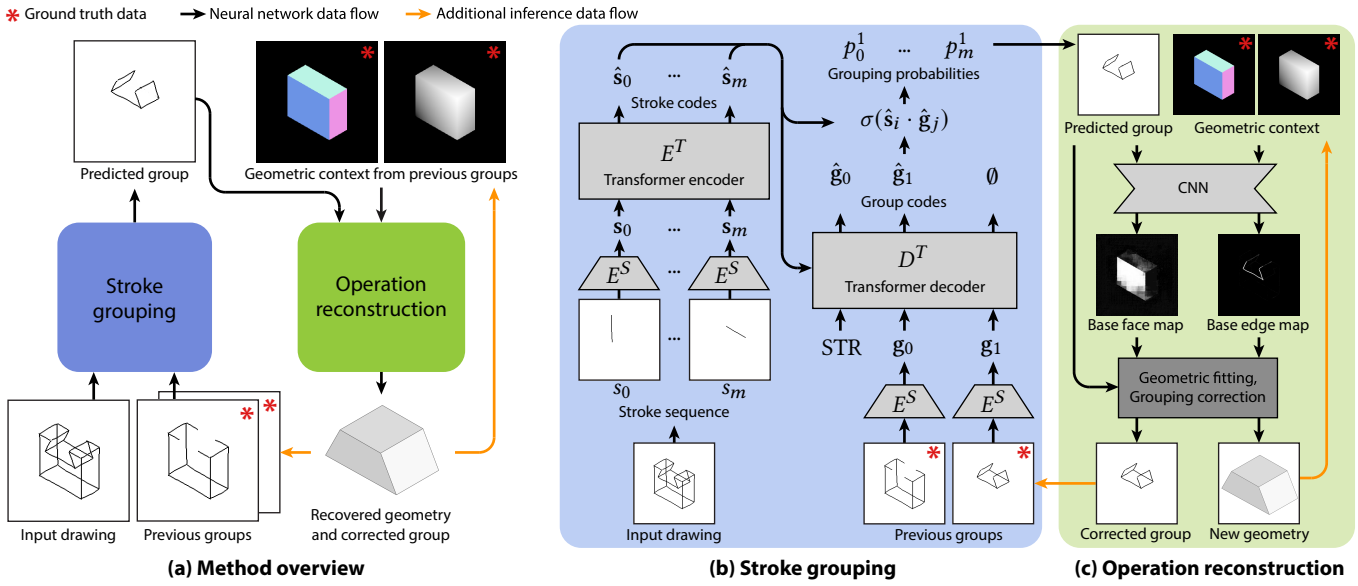


Fig. 3. **Overview.** Free2CAD converts a user drawing, presented as an ordered set of strokes, to a sequence of CAD commands (a). Our method runs in two main phases. First, in the *stroke grouping* phase (b), individual strokes s_i, encoded as $\mathbf{s}_i$, are processed by a Transformer, comprising of an encoder E^T and decoder D^T, to produce grouping probabilities p_i^j for each of the strokes in the context of the previous encoded group information $\mathbf{g}_j$. Second, in the *operation reconstruction* phase (c), the group probabilities are processed, conditioned on existing geometric context, to produce geometric primitives along with their parameters. The regressed geometric primitives are used to further correct the current grouping and the updated groups are passed back as context to the subsequent stroke grouping. Note that, in the grouping correction step, if the geometric fitting is unsatisfactory, current strokes along with the fitted primitives may be skipped (see Section 6). The two phases are trained using ground truth information, when available, to provide context information.

3 OVERVIEW

CAD modeling is difficult for novices because they often struggle to mentally decompose their target shape into a sequence of CAD commands. We aim at making CAD modeling accessible to a broader audience by letting users directly draw the target shape they have in mind, and leverage deep learning and geometric fitting to automatically decompose the drawing into parts that can be expressed as CAD commands.

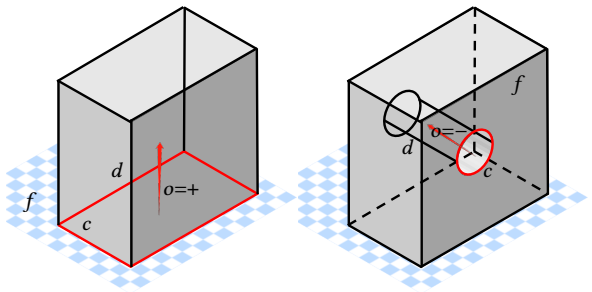


Fig. 4. **Primitives and operations.** Our prototype expresses the shape as a sequence of extrusion operations, where each operation adds or subtracts an extruded shape from a base shape. We parameterize an extrusion operation by the base face f on which it applies, the base curve c (polygon or circle) which defines the extrusion primitive shape, the offset distance d that defines the extrusion length, and whether the extruded primitive is an addition (o=+) or subtraction (o=−) to the base shape. To start, the base face f of the first operation is the ground plane and the add/subtract option is addition (o=+).

Formally, our system takes as input an ordered sequence of strokes $S = [s_i]$ that depicts a complex shape, and outputs a sequence of CAD commands $O = [o_j]$ which, when executed, reproduces well the depicted shape. Following recent work on data-driven CAD modeling [Li et al. 2020; Willis et al. 2021; Wu et al. 2021], we demonstrate our approach on a simplified CAD language that supports the positive or negative extrusion of polyhedral shapes and cylinders, as illustrated in Figure 4. While this language would not suffice to model complex shapes, for many everyday consumer products the simplified setting is sufficiently expressive and allows us to progress towards the ambitious goal of forming CAD models directly by sketching.

Given this problem definition, our key challenge is to group the input strokes into subsets $\mathcal{G}_j = \{s_i^j\}$, based on which we recover one CAD command o_j per group. Figure 3 details the main components of our proposed method.

In the first step, we design a sequence-to-sequence neural network to identify which strokes should belong to the same group (Sec. 4). This neural network selects one group of strokes at a time, yet processes consecutive groups together to share local information. We describe how to apply this neural network in a sliding window fashion to handle sequences of groups of arbitrary length (Sec. 4.3).

In the second step, we take the subset of strokes that forms each group and optimizes the parameters of the CAD operation to best

reproduce these strokes. This step is similar to the recent interactive sketch-based modeling system by Li et al. [2020], and we build on their methodology to relate strokes to CAD parameters (Sec. 5). Note that since the user draws in our system, camera information is always available.

Importantly, the two parts of our method are not separated; they are intimately connected during execution such that the fitted CAD commands provide geometric context to correct for errors in stroke grouping, and to fit subsequent operations (Sec. 6).

4 SEQUENTIAL GROUPING OF STROKES

The main challenge we face to create CAD models from complete drawings is to segment the drawing into groups of strokes, such that each group corresponds to a CAD command, and the resulting sequence of operations composes into the final shape. Given the sequential nature of both our input and output, we cast stroke grouping as a sequence-to-sequence translation task, which we implement using a *Transformer* architecture, popularized in natural language processing [Vaswani et al. 2017]. However, applying this architecture in our context opens several questions. How should we feed our drawings to such a sequential neural network? How should we model the output of the neural network to select strokes from an arbitrary number of groups? How can we scale the method to handle long sequences of groups? We answer these questions in the next sections, before describing how we fit CAD operations on the extracted groups of strokes.

4.1 Stroke Embedding

The original Transformer architecture was designed to process text, and relied on a learned embedding to encode each word as a vector of fixed size for subsequent processing. There are two popular options to encode drawings into such a compact, fixed-size representation. A first option is to encode the drawing using a CNN [Li et al. 2019] or a graph convolutional network [Yang et al. 2021], which allows the compact vector to capture information about the shape and spatial layout of the strokes. A second option is to treat the drawing as a sequence of stroke points [Ha and Eck 2018; Ribeiro et al. 2020], which allows the compact vector to capture information about drawing order. We combine these two options by treating the drawing as a sequence of bitmap strokes, and by encoding each stroke using a CNN.

Denoting the rasterized image of a stroke s_i as $\mathbf{I}(s_i) \in [0,1]^{256\times256}$, we pre-train a CNN auto-encoder architecture by minimizing the reconstruction loss

$$\left\| \mathbf{I}(s_i) - D^S\left(E^S\left(\mathbf{I}(s_i)\right)\right)\right\|^2,$$

where E^S and D^S denote the stroke encoder and decoder, respectively. We then discard the decoder and freeze the encoder to train the grouping Transformer, to which we feed the compact code $\mathbf{s}_i = E^S\left(\mathbf{I}(s_i)\right)$. Following the original Transformer architecture, we augment each code with a sinusoidal *positional encoding* [Vaswani et al. 2017], which in our context allows the network to reason about the relative ordering of strokes within the drawing sequence.

Since the Transformer groups the strokes in an auto-regressive fashion (see below), we also need a way to feed it an embedding of the strokes that have already been grouped. We do so by also training the CNN auto-encoder to reconstruct drawings composed of multiple strokes. Denoting $\mathcal{G}_j$ a group of strokes and $\mathbf{I}(\mathcal{G}_j)$ the corresponding image, we express the compact code of a group as $\mathbf{g}_j = E^S\left(\mathbf{I}(\mathcal{G}_j)\right)$.

4.2 Grouping Transformer

Given the sequence of input strokes, each encoded as a compact vector $\mathbf{s}_i$, the task of our Transformer neural network is to group strokes that represent the same CAD operation, one operation at a time. One option would be to express stroke grouping as a classification task, and train the Transformer to predict a sequence of group labels of same length as the input [Qi and Tan 2019]. But this option has several drawbacks: first, it would require knowing the number of groups in advance, and second, it would require generating the entire sequence of labels before attempting to assign a CAD operation to each group. Instead, we express stroke grouping as a *selection task* using a mechanism inspired by *Pointer Networks* [Vinyals et al. 2015].

The main intuition behind this mechanism is to train the Transformer encoder-decoder architecture to predict a compact code that is representative of the group of strokes to be selected, and compare this code to the code of each stroke to decide which one to keep. To do so, the Transformer encoder E^T embeds all input strokes into a latent space based on mutual attention, such that each stroke is now associated with a compact code $\hat{\mathbf{s}}_i = E^T(\mathbf{s}_i)$ that accounts for all other strokes in the sequence. The Transformer decoder D^T then takes as input the sequence of stroke codes $[\hat{\mathbf{s}}_i]$ and predicts the code of the current group, which we denote $\hat{\mathbf{g}}_j$. Given these two types of code, we express the probability that stroke s_i belongs to group $\mathcal{G}_j$ as

$$p_i^j := \sigma(\hat{\mathbf{s}}_i \cdot \hat{\mathbf{g}}_j),$$

where $\sigma(\cdot)$ is the sigmoid activation function, and $p_i^j > 0.5$ indicates that stroke s_i should be selected. Note that since our task requires selecting multiple elements at once, our formulation differs from the original Pointer Networks that rely on a softmax function to select one element at a time. Finally, we train the grouping Transformer to generate codes that satisfy our selection rule by minimizing the binary cross-entropy loss function:

$$\sum_{\mathcal{G}_j}\sum_{s_i}(\mathbb{1}_{s_i\in\mathcal{G}_j})log(p_i^j) + (1-\mathbb{1}_{s_i\in\mathcal{G}_j})log(1-p_i^j), \qquad (1)$$

where $\mathbb{1}$ is the indicator function.

Similarly to the original use of Transformer, where translated sentences are generated one word at a time, we apply the architecture auto-regressively to predict one group at a time. Formally, given a start token STR, the decoder D^T outputs a sequence of group codes $\hat{\mathbf{g}}_j$ inductively, *i.e.*, $\hat{\mathbf{g}}_j := D^T(\text{STR}, \mathbf{g}_0, \cdots, \mathbf{g}_{j-1}; [\hat{\mathbf{s}}_i])$, where the current group is predicted based on attention to the previous groups and all encoded strokes. In this formulation, we use the CNN encoder described in the previous section to compute the compact codes $\mathbf{g}_{0\ldots j-1}$ that represent the strokes selected in previous groups. At *training time*, we compute these codes using images of the ground truth groups; while at *test time*, we compute codes using images of the groups corrected by the geometric fitting procedure,

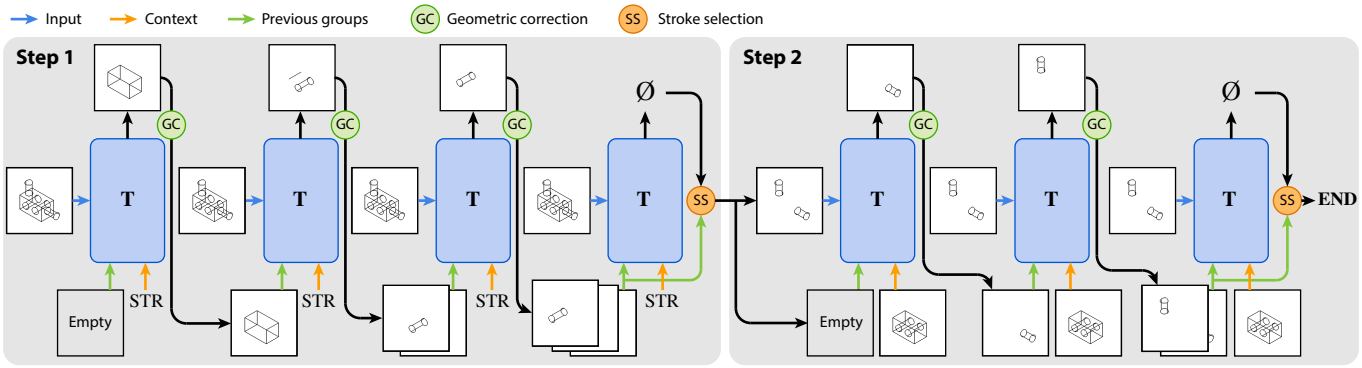


Fig. 5. **Processing long sequences.** In order to process long sequences of input strokes, we propose a scheme that runs in multiple Steps and, in each Step, processes a maximum of K groups. Specifically, we train Free2CAD (see Figure 3) to look at the current input sequence, process only K groups, and mark all the remaining strokes into a tail group. We then remove the strokes that have been successfully processed, and start the next Step with the remaining strokes. In this example, working with $K = 3$, the entire sequence of 28 input strokes is handled in two Steps outputting a total of 5 groups.

as described in Sec. 6. Finally, we indicate the end of the sequence via an empty group $G_\emptyset = \{\}$.

Group order ambiguity. Ordering the groups is an ambiguous task, because CAD commands can often be permuted yet produce the same outcome. To remove ambiguity, we order the CAD operations to agree with the occurrence of the corresponding strokes, *i.e.*, $\mathcal{G}_i \prec \mathcal{G}_j$ iff $s_0^i \prec s_0^j$ in the input drawing, where $\prec$ denotes precedence in a sequence. In other words, we assume a partial ordering of the operations in the input drawing, even though we allow users to start drawing a new operation before finishing the previous ones. See Sec. 8.5 for more discussions about the impact of this design on user sketching.

4.3 Processing Long Sequences of Groups

Equation 1 computes the loss function over all groups at once, which allows the Transformer to share information between groups. However, training such an architecture to handle a varying number of groups would require a very large dataset, where sequences composed of 1 to N groups would appear in equal proportions. An alternative option would be to only train the Transformer to select one group against all others, and apply the architecture iteratively until all strokes have been selected. But this scheme would prevent gradient back-propagation across groups, making the current group blind to the errors it might induce in subsequent groups. Our solution offers a balance between these two extremes. Specifically, observing that complex objects are often composed of locally independent parts, we adopt a sliding-window scheme that progresses over the sequence of groups by predicting K groups together, conditioned on the groups predicted so far, as shown in Fig. 5.

Concretely, given a sequence of $N > K$ groups, we train the Transformer to predict K groups $\mathcal{G}_{j\ldots(j+K-1)}$, along with a *tail group* that contains all strokes of groups $\mathcal{G}_{(j+K)\ldots N}$. We set j to an arbitrary position within the sequence, and we replace the start token STR by the CNN encoding of the previous groups $\mathbf{g}_{k<j}$ for $j \neq 0$. During inference, the tail group is recursively processed by the network until it becomes empty; meanwhile, the strokes selected in the earlier iterations are encoded to form the context $\mathbf{g}_{k<j}$.

As shown in Sec. 8.1, this sliding window scheme handles multiple groups robustly and achieves better performance than a naive training on long sequences because it breaks cluttered sketches into manageable sub-parts. We also experimented with different sizes of the sliding window and found that $K = 3$ offers a good trade-off between complexity of the architecture and prediction accuracy.

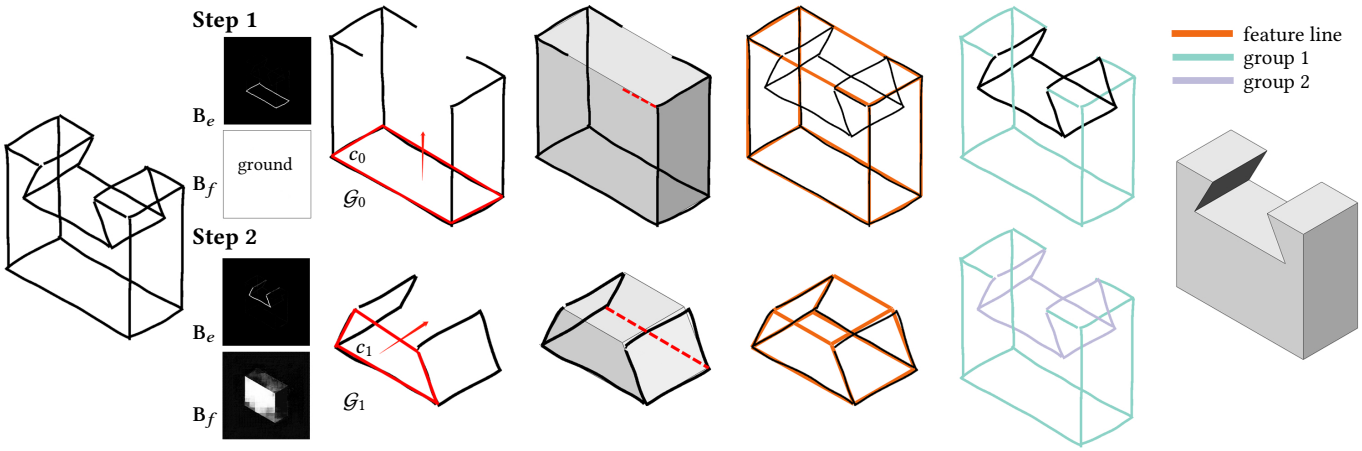
5 PER-GROUP CAD OPERATION RECONSTRUCTION

For each group of strokes $\mathcal{G}_j$ selected by the grouping Transformer, our next challenge is to recover a CAD operation o_j that reproduces well these strokes. This task is similar to the one faced by Li et al. [2020] in their interactive sketch-based modeling system, where they rely on a context-aware CNN architecture to recognize and fit CAD operations on individual drawing steps. However, Li et al. assume that users draw complete geometric primitives within each step, which are added or subtracted from the current shape to sequentially create complex objects. In contrast, since we take as input a single drawing of the final shape, individual groups often only represent an incomplete geometric primitive, as shown in Fig. 6. We now describe how we adapt their approach to handle this more complex scenario.

The extrusion operations we support are parametrized by a *base curve* that is extruded from a *base face* of the existing geometry, as detailed in Fig. 4. Given the shape M reconstructed from prior groups and the probabilities P_i^j of each stroke to belong to the current group $\mathcal{G}_j$, we proceed with the following steps to recover these parameters:

Contextual information. We first generate depth and normal maps $\mathbf{D}, \mathbf{N}$ of M. These maps provide contextual information to identify the strokes that form the base curve to be extruded, and the face of M from which the curve should be extruded. For the very first group $\mathcal{G}_0$, we render a ground plane as context.

CNN-based segmentation. Next, we rasterize all the strokes to form an image $\mathbf{I}(S)$, where we weight each stroke s_i by its probability P_i^j such that the strokes that are likely to belong to group $\mathcal{G}_j$ are more opaque. We adopted this soft blending strategy to preserve



(a) Input (b) Parameter fitting (c) Grouping correction (d) Final shape

Fig. 6. **Geometric correction.** Illustration of per-group operation reconstruction and grouping correction. Given the input strokes (a), the network regresses the segmentation maps (b) and predicts the grouped strokes $\mathcal{G}_0$ and $\mathcal{G}_1$ (c). The parameter fitting algorithm step then finds the fitted base shapes c_0 and c_1. Note here, the per-group sketch is frequently incomplete, as it depicts only those lines that remain in the final shape ($\mathcal{G}_1$ and c_1). After applying the extrusion operation, feature lines of the constructed shapes are used in the later grouping correction process (d), where full curve matching is applied first, and partial matching is followed to handle broken strokes. In this example, in Step 1, the small line (dotted red line) is erroneously *missed* in the initial group but reassigned to group 1 after the grouping correction step (i.e., the big cuboid); in Step 2, the back line (dotted red line) is erroneously *missed* in the initial grouping, but reassigned to group 2 after the grouping correction step (i.e., the prism). Finally, we get the resulting shape (d).

uncertainty from the grouping task. We concatenate this image with the contextual maps $\mathbf{D}, \mathbf{N}$ and process them with a CNN to predict an edge map $\mathbf{B}_e$ representing the base curve, and a segmentation mask $\mathbf{B}_f$ representing the base face. We use the same UNet neural network and the same loss as Li et al. [2020] for this task.

Base face and strokes selection. We use simple counting to identify which face $f \in M$ best covers the foreground pixels in $\mathbf{B}_f$, and which strokes $S' \subset S$ best cover the foreground pixels in $\mathbf{B}_e$.

Parameter fitting. We assume that the base curve is made of a single connected component, so we first discard the strokes that do not belong to the largest component of S'. We then back-project the strokes onto the 3D plane of f to obtain 3D curves. If the strokes form a closed loop, we fit either a circle or a polygon to the 3D curves to obtain the base shape. However, if the group represents an incomplete primitive, we complete the base shape by closing the loop with the boundary of the base face, as illustrated in Fig. 6. Finally, we construct the 3D primitive by extruding the base shape P along the normal direction $\mathbf{n}_f$ of f, applying line search to find the positive or negative offset d for which $P + d\mathbf{n}_f$ best re-projects on the remaining strokes $s_i \in \mathcal{G}_j \setminus S'$, where we measure the fitting error as the single-direction Chamfer distance from the strokes to the re-projected feature lines of the shape.

6 ITERATIVE EVALUATION WITH CORRECTION

Equipped with the ability to group strokes and to fit CAD operations, we process an input drawing by interleaving these two tasks, grouping and fitting one operation at a time. However, both of these tasks can produce erroneous results, which propagate to subsequent operations if not treated carefully. Our solution to this challenge is to exploit the complementary information provided by each task. On the one hand, we leverage the parametric CAD operation as a strong regularizer to identify and correct mislabeled strokes that do not align with the recovered 3D shape. On the other hand, we consider that a CAD operation has not been well recognized if too many strokes need to be corrected to make the group agree with the recovered shape.

Correcting erroneous groups. During training, we predict each new group $\mathcal{G}_j$ by feeding the Transformer decoder D^T with *ground-truth* preceding groups $\mathcal{G}_{0\ldots j-1}$, a strategy known as *teacher-forcing* in sequence-to-sequence learning. However, while this strategy stabilizes training, it does not reflect a realistic inference scenario where D^T is fed with groups *predicted* in the previous iteration. Since predicted groups are imperfect, the network accumulates errors and its performance degrades as it progresses through the sequence, an issue known as *exposure bias* [Mihaylova and Martins 2019].

To tackle this issue, during inference we perform an online correction of each group before it is used to predict the next group. Given the 3D shape produced by the recovered operation o_j, we project its feature lines in the drawing plane and attempt to match them to all strokes that have not been assigned to any of the previous groups, $s_i \notin \mathcal{G}_{0\ldots j-1}$. We first detect strokes that fully cover the feature lines by computing the bidirectional Chamfer distance between each stroke and each feature line, keeping the ones for which this distance is below a threshold $\epsilon_1 = 2\%$ of the diagonal length of the stroke's bounding box. We then identify strokes that only cover part of a feature line, which is common in the presence of subtractive extrusions. We detect these strokes by computing the Hausdorff distance from short strokes to long feature lines and keeping the ones for which this distance is below ϵ_1. However, this procedure might select multiple short strokes that belong to different parts of the object and yet are close to the feature line. When multiple such strokes overlap, we only keep the ones that are connected to

strokes that were identified to fully cover feature lines of the shape. Fig. 6 illustrates this correction on a typical prediction.

Detecting erroneous CAD operations. The CAD operations recovered by the procedure described in Sec. 5 are sensitive to the accuracy of the stroke groups, and to the accuracy of the base face and curve segmentation. We include several safeguards against erroneous operations. First, we detect cases where the base face or curve cannot be identified robustly from the predicted maps $\mathbf{B}_e$ or $\mathbf{B}_f$. This typically happens in ambiguous configurations, where the maps exhibit weak, blurry foregrounds from which no dominant faces or edges emerge. Second, we detect cases where the group predicted by the Transformer and the group corrected by the CAD operation differ by more than half of their strokes. In these cases, we do not include the CAD operation in the final shape, and we do not correct the predicted group before proceeding with the next group. While this strategy can result in incomplete shapes, it prevents catastrophic failures where erroneous operations degrade the recovery of subsequent ones.

7 IMPLEMENTATION

7.1 Dataset

As there is no existing dataset that contains paired CAD modeling sequences and corresponding sketches segmented according to CAD commands, we exploit the procedural nature of CAD models to synthesize a large dataset with the ground-truth labels needed to train our network.

Sequence generation. We generate each sequence of operations $[o_i]$ starting from the ground plane. For each subsequent operation, we randomize its base shape from the collection of N-gons and circles, where $N \in \{3, 4, 5\}$. The base face on which the operation is applied is always selected from the front-facing planes of the existing shape (viewpoint selection is discussed next), as otherwise an invisible base face provides ambiguous context for operation reconstruction. The other parameters of the operation, i.e., add/subtract and the offset distance, are then chosen randomly. Finally, we check the validity of the primitive shape constructed from the operation, to remove degenerate shapes that contain self-intersections, are out of the unit bounding box, or have too short projected strokes.

We synthesized sequences of up to 5 operations and balanced the occurrence of each sequence length with a ratio of 1:10:10:10:10. We denote splits of this dataset into sequences of specific lengths as $S1 \cdots S5$. In total, we have generated 82k valid sequences of operations, and randomly split them into training and testing sets by ratios of 80% and 20% respectively.

Sketch generation. Given a sequence of commands $[o_i]$, we execute them to obtain a corresponding sequence of intermediate shapes $[M_i]$. We extract the sketch from the final shape M_n as the collection of feature curves $S = \{s_j\}$, and assign each feature curve to the group corresponding to the command which first introduces it, i.e., $s_j \in \mathcal{G}_{k^*}$ such that $k^* = \min_k s_j \in M_k$. Examples of synthetic sketches are shown in Fig. 7.

We randomly shuffle the strokes to make our network robust to diverse stroke ordering (Fig. 2). However, to avoid group ordering ambiguity (Sec. 4.2), we ensure that $s_0^i \prec s_0^j$ for two groups $\mathcal{G}_i \prec \mathcal{G}_j$, where s_0^i denotes the first stroke in group $\mathcal{G}_i$.

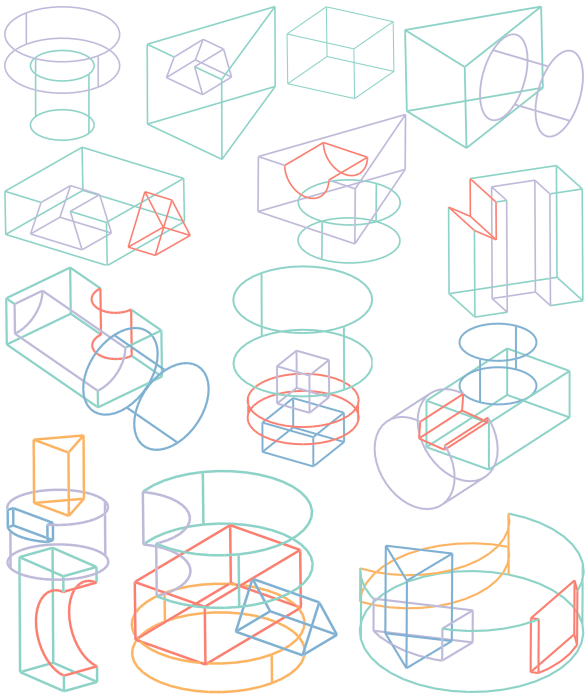


Fig. 7. **Synthetic training data.** Sketches synthesized by our automatic process (here shown without stroke perturbation). Different colors illustrate different groups. From top to bottom, we show random shapes produced by sequences of 1, 2, 3, 4, and 5 operations.

View selection and stroke rendering. Following the common practice of design sketching and sketch-based modeling systems, we assume that our input sketches are drawn under an *informative* perspective viewpoint, also known as 3/4 view, that shows the shape with minimal foreshortening on all sides. Therefore, when synthesizing the CAD operations and rendering the resulting models as sketches, we select a 3/4 view oriented towards the ground plane and uniformly sample 20 other viewpoints around it within 30 degrees variation, to simulate slight deviation of viewpoints in approximate sketches. Finally, we render all strokes using a similar style and perturbation scheme as in Li et al. [2020] to mimic hand-drawn inputs and make the system robust to inaccurate sketches (see supplemental material for more details).

Data augmentation for sliding window training. We also use our synthetic dataset for training the network with the sliding window scheme. To simulate the sliding window, we randomly skip several operations (including no skipping) for a given sequence. We compose the skipped operations to form a shape that serves as context, and train the network to predict the remaining operations. We applied this procedure on half of the original dataset to augment it to 123k sequences that we used to train the sliding window networks.

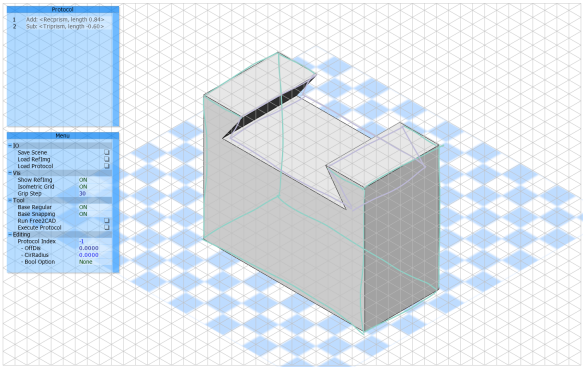


Fig. 8. **Free2CAD interface.** Interface of the prototype system implemented for Free2CAD. The UI features a sketching canvas with isometric grid to facilitate user sketching, the parametric CAD command list interpreted from the sketch, and editing tools for each command.

7.2 Network training

We implement our neural networks using Tensorflow; please refer to the supplemental material for network details. We split the network training into two phases. First, the grouping transformer network is trained until convergence, which takes 20 epochs and 9 days on a single GeForce GTX TITAN GPU. Second, based on the grouping output, we further train the operation reconstruction module, which takes 25 epochs and 6 days on the same GPU. We used the Adam solver [Kingma and Ba 2015] with fixed learning rate 10^{-4} and default parameters for the training.

We use the two-phase schedule because we found that training the two modules end-to-end to convergence takes much longer than training them separately, while the two strategies gave mostly the same final accuracy. Intuitively, the grouping task has full supervision and does not benefit from gradients from the operation reconstruction module; on the other hand, the operation reconstruction module depends on a well-trained grouping network to learn the mapping between strokes and segmentation maps.

7.3 Prototype Implementation

We have combined all the ingredients of our algorithm to develop a prototype sketch-based CAD modeling system. Typical interactive modeling sessions using the system can be found in the supplemental video.

User interface. A screenshot of the prototype interface is shown in Fig. 8. We overlay an isometric grid over the drawing canvas to help users draw under orthographic projection, a feature that participants of our study greatly appreciated. We also place an initial ground plane under a 3/4 view that serves as context for the first CAD operation reconstruction.

Once a shape is drawn and reconstructed, we allow users to rotate the shape and continue drawing from a novel viewpoint. We then run our method on this new drawing using the existing shape as context. This feature is particularly relevant for complex shapes that are difficult to draw from a single viewpoint, but we only used it for one result shown in this paper (Fig.9a). We facilitate 3D navigation by letting the user click on a face of the existing shape under the novel viewpoint, and automatically snap the camera such that this face becomes observed under a 3/4 view (see the accompany video for a demonstration).

A major strength of our approach is that it produces interpretable and editable sequences of CAD commands, which we leverage by allowing users to edit operation parameters as in traditional CAD software. Please refer to the accompanying video for such a parameter editing example. However, we did not use this feature for the results shown in the paper, nor for the user study so as to demonstrate the results of the main Free2CAD algorithm.

Regularization. In addition to the features mentioned above, we facilitate modeling of precise shapes by regularizing individual pen strokes, and by regularizing the recovered 3D shapes. We apply smoothing on the raw user strokes and we snap straight lines to the edges and vertices of the isometric grid. We snap the base shapes into their closest regular shapes, *e.g.*, right triangle, rectangle, pentagon, when this change remains within a small threshold. For subtraction operations, we snap the extrusion distance when the extruded face is close to an existing shape face. We also align pairs of adjacent primitives, *e.g.*, to align their centers or edges when close enough. Note that the regularization feature is optional and can be switched on/off in the interface.

8 RESULTS AND DISCUSSION

Using our modeling system, we have produced several CAD models with varying complexity. Typical examples are shown in Figs. 1 and 9, additional results can be seen in the supplemental material. We modeled all these results using a single drawing, except for Fig. 9a where we applied a 3D rotation to carve a hole on the bottom of the initial shape. To validate the effectiveness of our tool, we have conducted a user study (Sec. 8.4), a comprehensive ablation study (Sec. 8.1), and also compared our grouper with a recent semantic stroke segmentation algorithm (Sec. 8.3). Besides, we evaluated the generalization ability of our approach on the Fusion360 [Willis et al. 2021] dataset which contains real-world CAD sequences (Sec. 8.2). The accompanying video shows real-time modeling sessions using our system.

Runtime. All the experiments and tests are performed on a desktop PC with Intel(R) Core i9-9900 3.1GHz CPU and NVidia RTX 2070 Super GPU. Typically, the inference time of all networks is instantaneous, taking around 40ms, and the overall time for a modeling trial with three steps including the network inference and the following post-processing is around 1s.

8.1 Ablation study

There are several technical design choices of our algorithm, including the sliding window scheme for longer sequences and the geometric correction step. We validate these components by ablation tests evaluated on the grouping task. We report the grouping accuracy statistics separately on test datasets of $S2$ to $S5$ to see how the sequence length representing sketch complexity impacts performance. We have omitted results on $S1$ because sequences made of a single command are trivial to process for all settings.

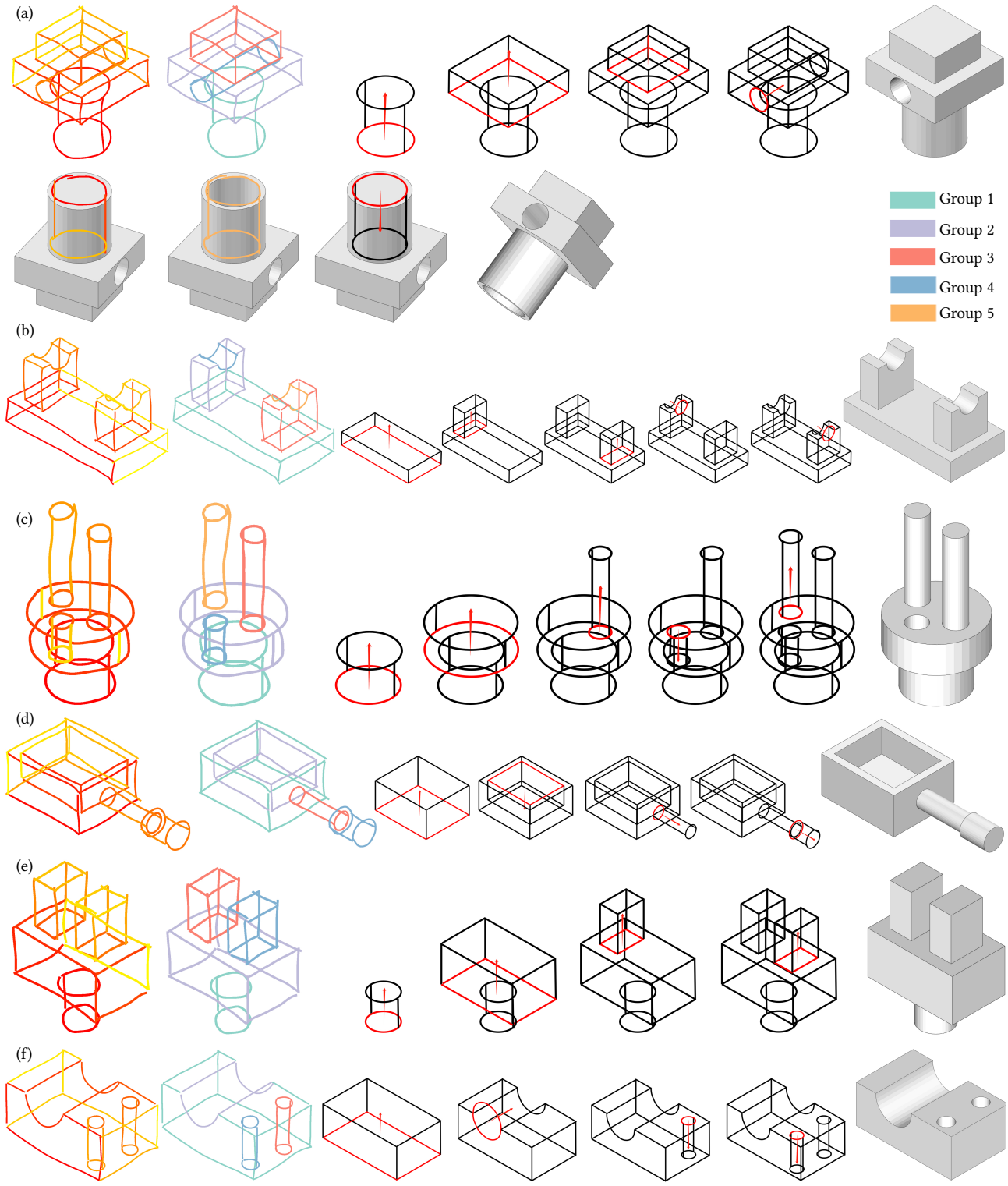


Fig. 9. **Result gallery.** We show a selection of modeling sessions using our method. In each case, the final model was created from a single view drawing (except example (a), where the last step is drawn from a second view) with 10-44 strokes that were respectively parsed into 4-5 groups, finally producing the corresponding inferred CAD models. All the models were drawn against an isometric grid background.

We have designed the following ablation settings:

- **Grouping network without sliding window**. We directly train a grouping network on the whole dataset containing sequences of 1 to 5 commands, denoted as *D5*.
- **Grouping network with sliding window of unit size**. For the sliding window scheme, a natural option is to predict one group at a time and leave the rest for future iterations. We denote this network as *Sliding(k=1)*.
- **Our design: a sliding window with three steps**. In our final design, we use a sliding window of width three and denote the resulting network as *Sliding(k=3)*. Furthermore, we tested several versions of this network to evaluate the impact of our geometric correction in comparison to a ground-truth correction.

The grouping accuracy of different settings are shown in Fig. 10. We define grouping accuracy as follows. Denote the ground-truth assignment of strokes to groups by $\mathbf{M}$, a binary matrix of dimension $S \times G$ where S is the number of strokes and G the number of groups, therefore $\mathbf{M}_{i,j} = 1$ if stroke s_i belongs to group $\mathcal{G}_j$ and zero otherwise; further denoting the predicted assignment matrix as $\mathbf{M}'$ of the same shape as $\mathbf{M}$, we compute grouping accuracy as

$$\frac{1}{S \cdot G} |\mathbf{M} - \mathbf{M}'|. \tag{2}$$

While it is expected that as the sequence length grows the performances of all settings degrade, there are contrasts among the settings.

Sliding window for longer sequences. By comparing *D5*, *Sliding(k=1, w/o correction)* and *Sliding(k=3, w/o correction)*, we observe that although *D5* is a grouping network directly trained on sequences of up to 5 commands, its performance is not as good as the network trained with a sliding window of 3 commands. We hypothesize that long sequences can be too cluttered and ambiguous to parse; while breaking long sequences into chunks of length 3 makes the grouping task easier to learn. We also observe that *Sliding(k=1, w/o correction)* performs the worst, which confirms that when limited to single group prediction, the network cannot learn to leverage the mutual information among consecutive groups.

Geometric label correction. The *Sliding(k=3, GT correction)* setting serves as an upper bound of sliding window settings, as ground-truth groups are sent to the decoder to predict the next group. Our final setting *Sliding(k=3, correction)*, used in real applications, instead uses online geometric correction based on the reconstructed primitives. Fig. 10 reveals that our final setting improves significantly over the version without correction, *i.e.*, *Sliding(k=3, w/o correction)*, and comes closer to the upper bound setting with ground-truth correction. This comparison demonstrates the efficiency of our sliding window scheme with online group correction.

To better understand the grouping accuracy, we have further evaluated the fraction of models that are successfully reconstructed on *S2* to *S5* using our full pipeline, where we consider that a model is successful if it has the correct topology (same number of operations) and geometry (low Chamfer distance). If we use tight topological and geometric tolerance, the ratios are 84%, 70%, 54%, and 28%, respectively. Note that these success rates are biased by the strictness

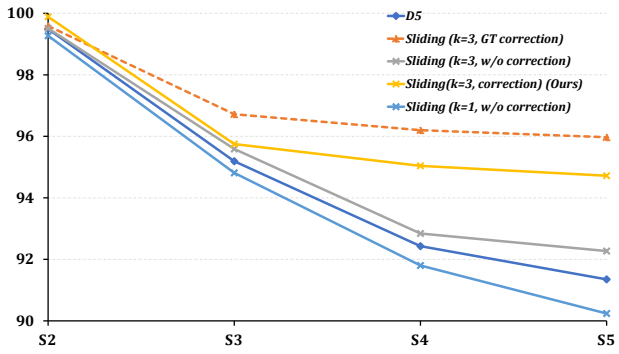


Fig. 10. Statistical evaluation of different grouping network configurations.

of metrics. In fact, we generally obtain reconstructions with decent overall shapes even for low accuracy samples; see the examples in Figs. 9 and 11, as well as more results in the supplemental material.

8.2 Evaluation on Fusion360 dataset

Recently, several CAD modeling datasets containing real user creations have been released [Koch et al. 2019; Willis et al. 2021]. While these dataset do not include sketches, we can leverage their CAD sequences to test the ability of our method to generalize to real shapes. To this end, we curated a representative subset (294 models) of Fusion360 [Willis et al. 2021] and cleaned and converted its command sequences to extrusion operations compatible with our system. We then synthesized strokes depicting the final shapes to serve as input for our method. To our knowledge, this is the first attempt to evaluate a sketch-based modeling system on such a large, out-of-training dataset.

Fig. 11 shows representative results of this experiment, where the shapes recovered by Free2CAD closely match ground-truth, even though the predicted sequences of operations often differ from the original Fusion360 sequences, which can contain operations we do not support, such as extrusions of multiple base shapes at once.

Fig. 12 provides more quantitative results by plotting several statistics as a function of the sequence length. In addition to the frequency of different command sequence lengths, the grouping accuracy as defined in Eq. (2) and the chamfer distance between our reconstructed shapes and the shapes from Fusion360, we also compute the ratio of strokes that are properly assigned to operations. As strokes can be left unassigned if its corresponding operation is not recognized or not properly reconstructed (see Sec. 5), the ratio of explained strokes also reflects how well our system parses the sketches. To measure the stroke grouping accuracy, we obtained ground truth sequences by converting the original sequences to the operations we support.

Fig. 12 shows that as the number of operations increases, the accuracy of our method progressively reduces, as measured on the grouped strokes as well as on the Chamfer distance between the reconstructed shape and ground-truth. The domain gap between our synthetic data and real CAD models is stronger on long sequences (5 operations and more), for which our method could only explain half of the strokes. In practice, we observed that our method often

succeeds to reconstruct the overall shape of the model from the first strokes, but struggles with details captured by later strokes, which explains why the Chamfer distance remains low even for sequences that are partly reconstructed. See the inset of two examples in Fig. 12(c) for the matching of specific values and visual differences. Nevertheless, our method achieves high accuracy on shorter sequences, which forms a large part of the Fusion360 dataset. The conversion procedure and additional results on the Fusion360 test set can be found in the supplemental material.

8.3 Comparison

To the best of our knowledge, we are the first to do geometric stroke grouping for CAD shape modeling. The most related task is semantic segmentation of freehand sketches; we compare our method to a state-of-the-art method developed for this task. From the perspective of sketch-based CAD modeling through command parsing, we also compare our method with Sketch2CAD [Li et al. 2020] that however assumes that a sketch represents a single command.

Sketch semantic segmentation. Our task is related to semantic segmentation of freehand sketches, but also has key differences. In semantic segmentation, labels are predefined, *e.g.*, a character has four semantic parts - head, body, arm and leg, and the task is to predict a part label for each stroke, so that strokes with the same label are grouped together. Moreover, semantic segmentation algorithms are usually applied on planar abstract drawings of objects rather than perspective drawings of 3D shapes.

Despite these differences, we trained the state-of-the-art segmentation approach SketchGNN [Yang et al. 2021] to segment an input sketch into three groups, each corresponding to a CAD command in a sequence. For this comparison, we trained both SketchGNN and our grouping network on a dataset consisting of $S1$ to $S3$. We did not train our network with the sliding window scheme, as the sequence length is at most three in this experiment.

The results are reported in Table 1, where stroke accuracy (denoted as *SAcc*) is computed as the percentage of strokes that are correctly assigned to corresponding groups, following the common practice of sketch segmentation tasks. Our grouping network performs significantly better than the default SketchGNN on this task. Since the stroke order is not used in SketchGNN but encoded in our sequence input, we further augment strokes fed to SketchGNN with an additional channel that contains the order information, which we denote as SketchGNN+. As shown in Table 1, while SketchGNN+ performs better than without stroke order input, it is still far less accurate than our network. We hypothesize that since SketchGNN uses a graph convolutional network defined over proximate strokes of the sketch, it struggles with perspective drawings where strokes that belong to different operations frequently overlap. In contrast, our approach benefits from the ability of the Transformer network to reason about long-range interactions between strokes.

Sketch2CAD. Aiming at a similar sketch-based CAD modeling task, Sketch2CAD [Li et al. 2020] adopts a sequential workflow to model complex shapes part-by-part from several views, which requires the user to decompose the target shape into CAD commands mentally and input the sketch of each command sequentially. Fig. 13 illustrates this fundamental difference, where Sketch2CAD attempts to recognize a single CAD operation from a drawing that would require two operations, and as a result fails to properly fit the operation to the drawn lines. In comparison, Free2CAD can successfully parse the complete drawing consisting of multiple operations and faithfully reconstruct the intended shape. While applying Sketch2CAD on the ground-truth groups of strokes one-by-one may work, our core contribution is to automate the grouping task.

Table 1. **Grouping accuracy.** Comparison with SketchGNN [Yang et al. 2021] on the stroke grouping task. Since the original SketchGNN, the state-of-the-art in semantic sketch grouping, does not use stoke drawing ordering, we also tested an augmented version (SketchGNN+) which had access to stoke drawing information. Ours performed better.

	SketchGNN	SketchGNN+	Our Grouper
$SAcc(\%)$	79.86	84.47	93.05

8.4 User study

To evaluate the expressiveness and accessibility of our modeling system, we invited 5 novice users without 3D modeling expertise for a study. The users are first given a 15-20min tutorial about the modeling system, where we present sample sketches and corresponding result shapes and discuss the drawing interface. After a short practice session where the users can freely try the prototype system, the users are asked to model two reference shapes (green in Fig. 14) that can be viewed in full 3D through a standalone viewer, by sketching in one view only. Note that the users are not required to reproduce the exact dimensions of the reference objects but are allowed to deviate from them for casual exploration.

We find that within 15 minutes[1] all the users can successfully sketch proper shapes. The drawings and result shapes obtained by the five users are shown in Fig. 14, where the strokes are again color-coded by their ordering within the sketches. As is expected, while the overall scheme still follows a coarse-to-fine approach where larger parts are drawn first as anchors for smaller parts, the users use very different drawing sequences for the same reference object, which our system handles robustly. Meanwhile, the users find it easy to conceive the sketches as they intuitively capture the feature lines of the target shapes; some expressed surprise that such line drawings can be instantly turned into complete shapes.

The users commented on the convenience provided by isometric grid for referencing and the regularization of line drawing, which helped them to draw relatively straight lines despite their lack of drawing experience (see also the line quality in Fig. 14). However, to draw curves and circles is a bit harder, as shown by e.g., the distorted curves of P5 for the second object. Nevertheless, our system can robustly parse such strokes and fit operation parameters.

8.5 Stroke drawing order

As noted in Sec. 4.2, our system is designed to process diverse drawing orders of the sketch depicting the final shape, albeit with

[1]We could not run in-person sessions due to COVID-related restrictions, thus due to internet delay, users had to wait for 1-2s before each stroke was displayed on their end, which slowed down the sketching sessions (see supplemental for detailed records).

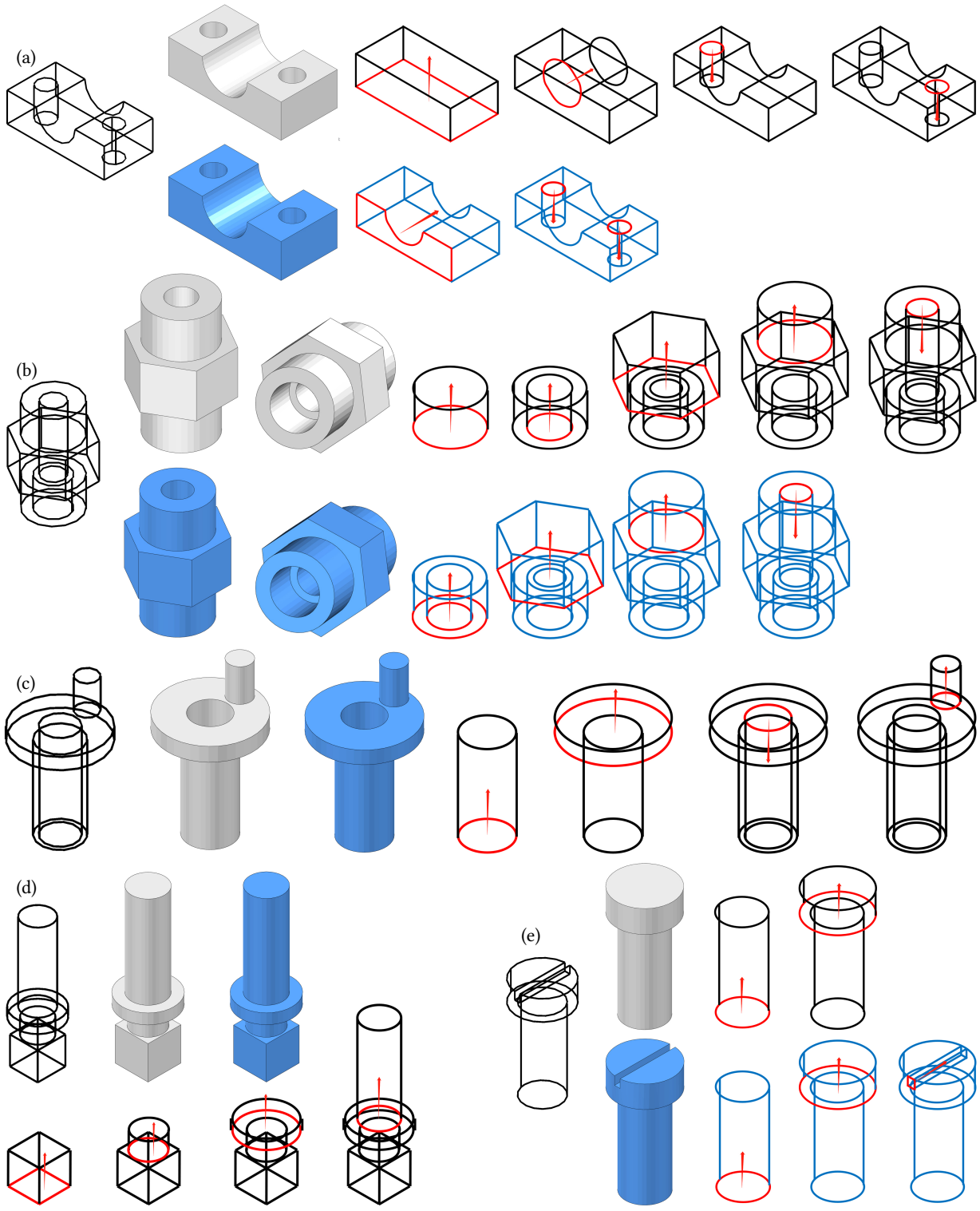


Fig. 11. **Evaluation on Fusion360.** We quantitatively evaluated Free2CAD on 294 models from the Fusion360 CAD dataset that were automatically converted to fixed view drawings. Primitives were drawn in the order they appeared in the CAD descriptions. While our method performed satisfactorily in recovering the 3D shapes, the sequence of operations were sometimes((a), (b), (e)) different as CAD commands only describe a partial ordering. In example (e), ours failed to recover the groove of the bolthead. In black and red, we show our recovered CAD command sequence; if different, in blue, we show Fusion360's ground truth.

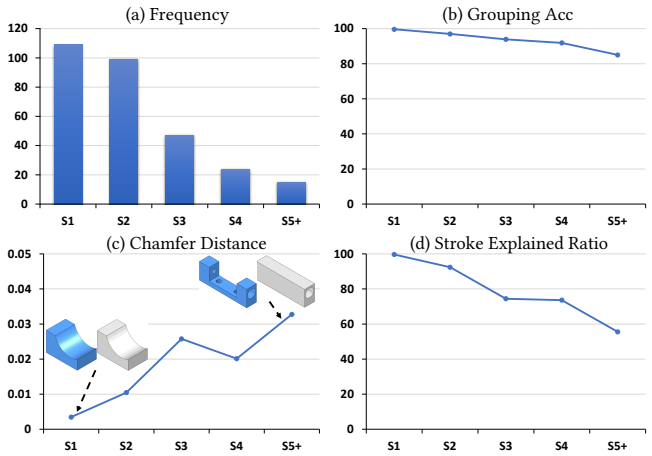


Fig. 12. **Quantitative evaluation on Fusion360.** Statistical evaluation on 294 models from the Fusion360 dataset. Inset figures provide visual examples, which use the same color coding as in Fig. 11. Please refer to texts for details.

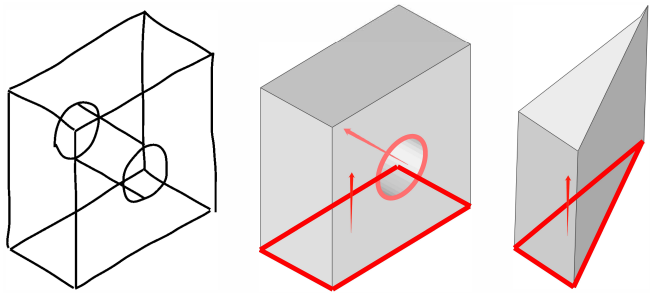


Fig. 13. **Comparison.** Given a sketch sequence (a), our Free2CAD can infer multiple CAD commands in one go (b). In comparison (c), Sketch2CAD [Li et al. 2020], which expects and interprets one primitive/operation at a time, wrongly infers a prism being extruded from a triangular base.

a weak assumption that the first strokes of all groups are ordered in the same precedence as the ordering of recovered commands. This weak assumption generally allows our algorithm to handle a large combination of possible sketch sequences with desirable CAD operations reconstructed, as shown in Fig. 15(a), where the drawings from left to right, right to left, top to bottom or bottom to top all produce the same resulting shape. However, under rare cases where the drawing order does not correspond to a feasible composition of CAD operations, the system may fail to parse or reconstruct the operations (e.g., a small subtracted part is drawn before the part which it is subtracted from, or the first operation is not anchored on the ground). Fig. 15(b) illustrates such as failure where the subtracted part is drawn first.

This weak ordering imposes constraints on the user sketching. Nevertheless, participants in our user study (Sec. 8.4) adopted this ordering because it corresponds to a coarse-to-fine creation process.

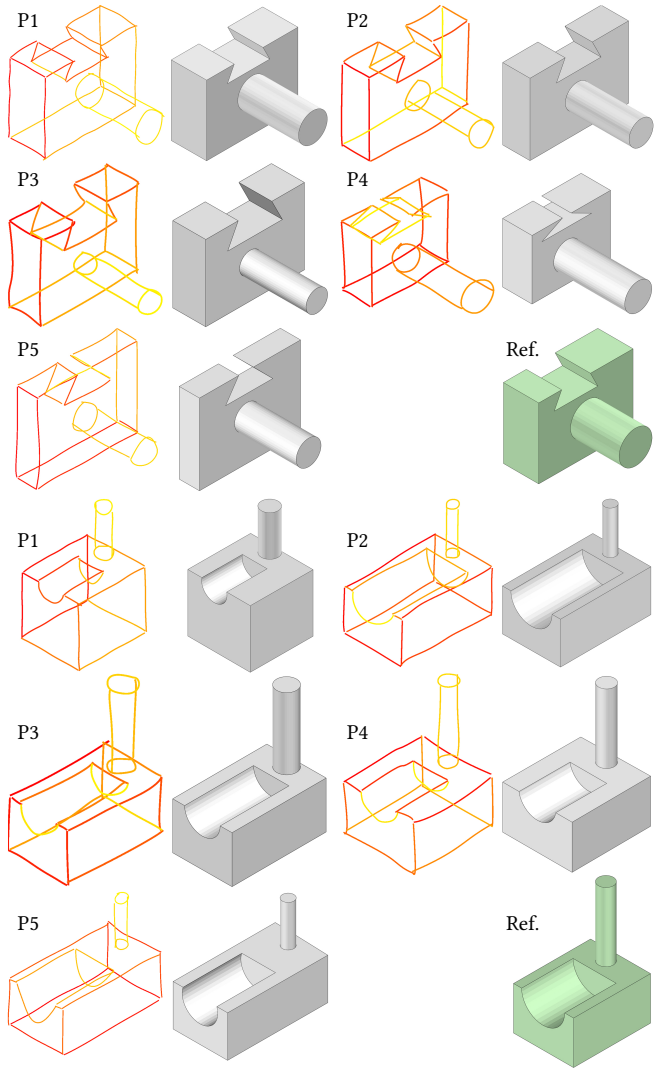
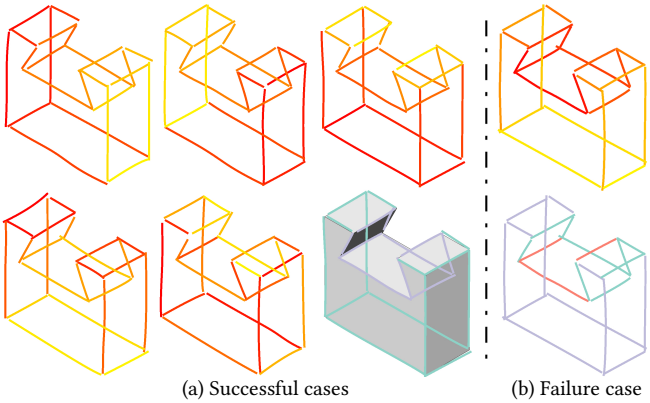


Fig. 14. **User study.** Users were asked to model two objects (Ref.), shown in green, using Free2CAD. Isometricgrid reference was turned on. All the users succeeded in modeling with our system, although there were variations in object dimensions and proportions. Similar to the pilot study findings, users sketched in different orders. For example, for the camera, P1 drew the lens at the end, while P4 sketched the lens before the flash socket; for the grooved-chimney, P1 drew the groove earlier compared to all the others.

9 CONCLUSION AND FUTURE WORK

Planning how to construct a 3D shape using CAD operations is a very challenging task. In this paper, we have explored the potential of machine learning and geometry processing to assist novice users in this task. We have presented Free2CAD that parses an ordered sequence of pen strokes into a sequence of CAD commands which, when executed, recreates the input sketch from the known camera view. At the core of our method is a novel stroke grouping task learned by a sequence-to-sequence neural network, along with a geometric fitting procedure that provides corrections and geometric



(a) Successful cases (b) Failure case

Fig. 15. **Robustness to stroke ordering.** Free2CAD is trained to handle different ordering of strokes. We can successfully handle permutations of stroke orderings, as long as the objects are drawn roughly in a coarse-to-fine order (see Pilot study in Figure 2). Our method can fail when this assumption is broken, as shown on the right where the prism-wedge is entirely drawn before the main body of the object.

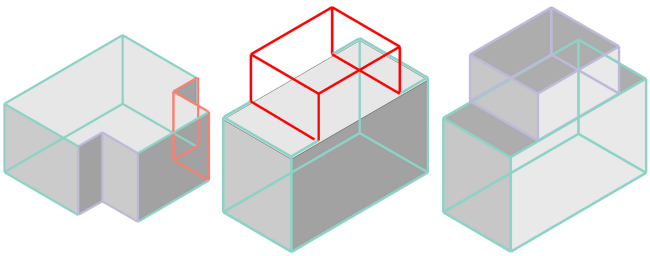
context in the autoregressive inference loop. We also introduce a sliding window scheme to translate long drawing sequences into multiple CAD commands. Our extensive evaluation demonstrates the advantage of the proposed setup on stroke grouping accuracy, as well as its generalization to real-world data despite being trained on synthetic CAD sequences. This ability to group strokes into CAD operations is the key enabler of the first end-to-end system that produces sequences of CAD commands directly from user drawings.

The proposed system has a set of assumptions and limitations that we propose to address in the future work.

Limited primitive/operation types. We demonstrated the core concepts of stroke grouping and CAD operation reconstruction using a restricted set of primitives (cuboids and cylinders) and operations (extrude and subtract perpendicularly to flat surfaces). While the stroke grouping network could be trained with a dataset that contains other primitives and operations, the challenge resides in recognizing more diverse operations (e.g., taper and bevel) at the operation reconstruction stage, as well as in implementing the corresponding robust parameter fitting from the predicted base and edge probability maps.

Limited drawing style. Our prototype primarily targets novice users and assumes that only the feature lines of the object are drawn (both visible and hidden lines). An exciting direction for future work would be to train our system on synthetic drawings that emulate the style of professional designers, and test it on real-world design sketching sequences [Gryaditskaya et al. 2019]. But professional designers employ specific *construction lines*, for which non-photorealistic rendering algorithms do not exist yet. In addition, real-world sketches exhibit more diverse viewpoints than our 3/4 orthographic view, they include over-sketched and over-shot strokes, and they depict CAD operations we do not support.

Ordering of strokes. Motivated by our formative pilot study, and by ordering principles described in sketching books [Dodson 1990; Edwards 1979; Steur 2011] and observed in recent drawing datasets



(a) (b) (c)

Fig. 16. **Limitations.** In case of objects with aligned faces, our approach's success may depend on the orientation of the object. For example, a T-shape can be modeled as two smaller cuboids subtracted from a bigger base cuboid (a); the same object is incorrectly inferred in the vertical orientation (b); however, we can successfully infer the shape when it is depicted as a decomposition of two convex objects, i.e., two stacked cuboids (c).

[Wang et al. 2021], we designed Free2CAD to handle drawings with a weak ordering among the strokes (i.e., largely coarse-to-fine modeling). Specifically, since our method relies on progressive geometric context to facilitate grouping and primitive/operation finding, some shapes may be successfully handled in one orientation but not in others. For example, in Figure 16, the T-shape is successfully recovered when drawn in the horizontal orientation (16a), rather in the vertical one (16b). This also relates to an inherent ambiguity of CAD commands — a sequence of CAD commands maps to a unique 3D shape but not the other way round. For example, the same T-shape is successfully parsed if it is drawn as two stacked cuboids (16c). In the context of manufacturing, however, one decomposition is more desirable than the other: two cuboids may be easier to fuse together, rather than having to subtract two cuboids from a larger cuboid. Defining a measure of the *manufacturability* of an inferred CAD sequence would greatly benefit generative modeling tasks like ours.

Exposure bias. We trained the Transformer using ground truth data for geometric context and stroke correction. In our tests, the geometry reconstructed from the inferred operations were sufficient to enable generalization from synthetic to real data. However, a more robust approach would be to actually reduce the exposure bias. Recent efforts [He et al. 2021] on robust decoders can help in this goal but we need to further account for the unique geometric context available in our setup.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their valuable suggestions, the user evaluation participants, Jian Shi, Yuxiao Guo and team remembers of both GraphDeco (INRIA) and SGP (UCL) groups for the valuable discussions, and Julien Philip, George Drettakis for proofreading earlier drafts of the paper. AB was supported by ERC Starting Grant D3 (ERC-2016-STG 714221); CL and NM were supported by European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 956585, ERC Grant (SmartGeometry 335373), and gifts from Autodesk and Adobe. NM thanks Tuhin for introducing him to the isometric grids.

REFERENCES

Autodesk. 2019. *TinkerCAD.* https://www.tinkercad.com/

Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. 2008. ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proc. UIST*. ACM, 151–160.

Suresh K. Bhavnani, Bonnie E. John, and Ulrich Flemming. 1999. The Strategic Use of CAD: An Empirically Inspired, Theory-Based Course. In *Proc. of the ACM SIGCHI Conference on Human Factors in Computing Systems*. 183–190.

Alexandra Bonnici, Alican Akman, Gabriel Calleja, Kenneth P Camilleri, Patrick Fehling, Alfredo Ferreira, Florian Hermuth, Johann Habakuk Israel, Tom Landwehr, Juncheng Liu, et al. 2019. Sketch-based interaction and modeling: where do we stand? *Artificial intelligence for engineering design analysis and manufacturing* (2019), 1–19.

Yu Chen, Jianzhuang Liu, and Xiaoou Tang. 2007. A Divide-and-Conquer Approach to 3D Object Reconstruction from Line Drawings. In *ICCV*.

Ivan Chester. 2006. Teaching for CAD expertise. *International Journal of Technology and Design Education* 17 (2006), 23–35.

Bert Dodson. 1990. *Keys to drawing.* Penguin.

Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. 2018. InverseCSG: Automatic Conversion of 3D Models to CSG Trees. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia* 37, 6 (2018).

Betty Edwards. 1979. *Drawing on the Right Side of the Brain.* Penguin.

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. 2019. Write, Execute, Assess: Program Synthesis with a REPL. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 32.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. 2018. Learning to infer graphics programs from hand-drawn images. In *Advances in neural information processing systems*. 6059–6068.

Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. 2021. Computer-aided design as language. *Advances in Neural Information Processing Systems* 34 (2021).

Yotam Gingold, Takeo Igarashi, and Denis Zorin. 2009. Structured Annotations for 2D-to-3D Modeling. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 28, 5 (2009).

Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. 2020. Lifting Freehand Concept Sketches into 3D. *ACM Transactions on Graphics* (2020).

Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Fredo Durand, and Adrien Bousseau. 2019. OpenSketch: A Richly-Annotated Dataset of Product Design Sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* (2019).

David Ha and Douglas Eck. 2018. A Neural Representation of Sketch Drawings. In *International Conference on Learning Representations (ICLR)*.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. 2021. Masked Autoencoders Are Scalable Vision Learners. *CoRR* abs/2111.06377 (2021).

Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-sketch: Output-directed programming for SVG. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 281–292.

Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. 2016. Shape Synthesis from Sketches via Procedural Models and Convolutional Networks. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 22, 10 (2016), 1.

R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2020. ShapeAssembly: Learning to Generate Programs for 3D Shape Structure Synthesis. *ACM Transactions on Graphics (Proc. Siggraph Asia)* 39, 6 (2020), Article 234.

Joaquim A Jorge, Nelson F Silva, and Tiago D Cardoso. 2003. GIDeS++: A Rapid Prototyping Tool for Mould Design. In *Proceedings of the Rapid Product Development Event RDP*.

Kacper Kania, Maciej Zieba, and Tomasz Kajdanowicz. 2020. UCSG-NET- Unsupervised Discovering of Constructive Solid Geometry Tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 33.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. ABC: A Big CAD Model Dataset For Geometric Deep Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J Mitra. 2020. Sketch2cad: Sequential cad modeling by sketching in context. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 39, 6 (2020), 1–14.

Changjian Li, Hao Pan, Yang Liu, Alla Sheffer, and Wenping Wang. 2017. BendSketch: Modeling Freeform Surfaces Through 2D Sketching. *ACM Trans. Graph. (Proc. SIGGRAPH)* 36, 4 (2017).

Ke Li, Kaiyue Pang, Jifei Song, Yi-Zhe Song, Tao Xiang, Timothy M Hospedales, and Honggang Zhang. 2018. Universal sketch perceptual grouping. In *Proceedings of the european conference on computer vision (ECCV)*. 582–597.

Lei Li, Hongbo Fu, and Chiew-Lan Tai. 2019. Fast Sketch Segmentation and Labeling With Deep Learning. *IEEE Computer Graphics and Applications* 39, 2 (2019), 38–51.

H Lipson and M Shpitalni. 1996. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-Aided Design* 28, 8 (1996), 651 – 663.

Tsvetomila Mihaylova and André FT Martins. 2019. Scheduled sampling for transformers. *arXiv preprint arXiv:1906.07651* (2019).

Chandrakana Nandi, James R. Wilcox, Pavel Panchekha, Taylor Blau, Dan Grossman, and Zachary Tatlock. 2018. Functional Programming for Compiling and Decompiling Computer-aided Design. *Proceedings of the ACM on Programming Languages* 2, ICFP (2018), 99:1–99:31.

Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. 2016. Interactive Sketching of Urban Procedural Models. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)* (2016).

GüNay Orbay and Levent Burak Kara. 2012. Sketch-based surface design using malleable curve networks. *Comput. Graph. Forum* 36, 8 (2012).

Wamiq Reyaz Para, Shariq Farooq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas Guibas, and Peter Wonka. 2021. SketchGen: Generating Constrained CAD Sketches. arXiv:2106.02711 [cs.LG]

Yonggang Qi and Zheng-Hua Tan. 2019. SketchSegNet+: An End-to-End Learning of RNN for Multi-Class Sketch Semantic Segmentation. *IEEE Access* 7 (2019), 102717–102726.

Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. 2020. Sketchformer: Transformer-based representation for sketched structure. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. 2009. Analytic drawing of 3D scaffolds. In *ACM transactions on graphics (TOG)*, Vol. 28. 149.

Rosália G. Schneider and Tinne Tuytelaars. 2016. Example-Based Sketch Segmentation and Labeling Using CRFs. *ACM Trans. Graph.* 35, 5, Article 151 (July 2016), 9 pages.

Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. 2021. Vitruvion: A Generative Model of Parametric CAD Sketches. *arXiv preprint arXiv:2109.14124* (2021).

Tianjia Shao, Wilmot Li, Kun Zhou, Weiwei Xu, Baining Guo, and Niloy J. Mitra. 2013. Interpreting Concept Sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 32, 4 (2013), 10 pages.

Shapr3D. 2016. *Shapr3D.* https://www.shapr3d.com/

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. 2018. CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. 2013. Geosemantic snapping for sketch-based modeling. In *Computer graphics forum*, Vol. 32. Wiley Online Library, 245–253.

Roselien Steur. 2011. *Sketching: The Basics.* BIS.

Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. 2019. Learning to Infer and Execute 3D Shape Programs. In *International Conference on Learning Representations*.

Trimble. 2019. *SketchUp.* https://www.sketchup.com/

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Advances in Neural Information Processing Systems*, Vol. 28. Curran Associates, Inc.

Zeyu Wang, Sherry Qiu, Nicole Feng, Holly Rushmeier, Leonard McMillan, and Julie Dorsey. 2021. Tracing versus freehand for evaluating computer-generated drawings. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–12.

Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. 2021. Fusion 360 Gallery: A Dataset and Environment for Programmatic CAD Construction from Human Design Sequences. *ACM Transactions on Graphics (TOG)* 40, 4 (2021).

Q. Wu, K. Xu, and J. Wang. 2018. Constructing 3D CSG Models from 3D Raw Point Clouds. *Computer Graphics Forum* 37, 5 (2018).

Rundi Wu, Chang Xiao, and Changxi Zheng. 2021. DeepCAD: A Deep Generative Network for Computer-Aided Design Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 6772–6782.

Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. 2014. True2Form: 3D curve networks from 2D sketches via selective regularization. *ACM Transactions on Graphics* 33, 4 (2014).

Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl D. D. Willis, and Daniel Ritchie. 2021. Inferring CAD Modeling Sequences Using Zone Graphs. In *CVPR*.

Lumin Yang, Jiajie Zhuang, Hongbo Fu, Xiangzhi Wei, Kun Zhou, and Youyi Zheng. 2021. SketchGNN: Semantic Sketch Segmentation with Graph Neural Networks. *ACM Transactions on Graphics* 40, 3, Article 28 (Aug. 2021), 13 pages.

Fenggen Yu, Zhiqin Chen, Manyi Li, Aditya Sanghi, Hooman Shayani, Ali Mahdavi-Amiri, and Hao Zhang. 2021. CAPRI-Net: Learning Compact CAD Shapes with Adaptive Primitive Assembly. arXiv:2104.05652 [cs.CV]

Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. 1996. SKETCH: An Interface for Sketching 3D Scenes. In *Proceedings of SIGGRAPH (Computer Graphics Proceedings, Annual Conference Series)*. 163–170.