PlankAssembly: Robust 3D Reconstruction from Three Orthographic Views with Learnt Shape Programs

Wentao Hu[1,2]†* Jia Zheng[3]* Zixin Zhang[4]* Xiaojun Yuan[4] Jian Yin[1,2] Zihan Zhou[3]
[1]Sun Yat-Sen University [2]Guangdong Key Laboratory of Big Data Analysis and Processing
[3]Manycore Tech Inc. [4]University of Electronic Science and Technology of China
https://manycore-research.github.io/PlankAssembly

Abstract

In this paper, we develop a new method to automatically convert 2D line drawings from three orthographic views into 3D CAD models. Existing methods for this problem reconstruct 3D models by back-projecting the 2D observations into 3D space while maintaining explicit correspondence between the input and output. Such methods are sensitive to errors and noises in the input, thus often fail in practice where the input drawings created by human designers are imperfect. To overcome this difficulty, we leverage the attention mechanism in a Transformer-based sequence generation model to learn flexible mappings between the input and output. Further, we design shape programs which are suitable for generating the objects of interest to boost the reconstruction accuracy and facilitate CAD modeling applications. Experiments on a new benchmark dataset show that our method significantly outperforms existing ones when the inputs are noisy or incomplete.

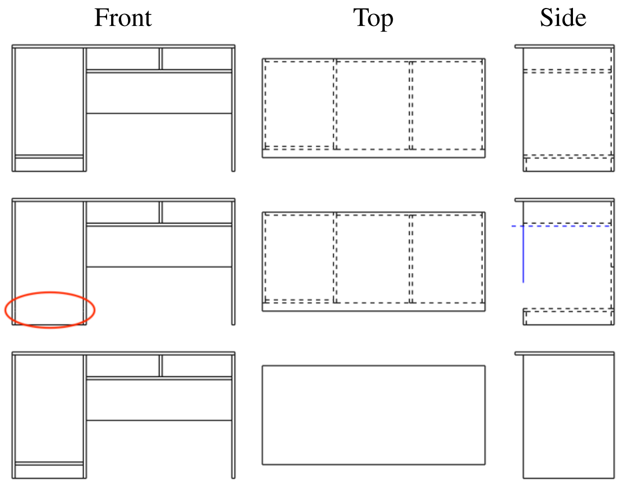

Figure 1. An illustration of various input drawings. **From top to bottom**: clean inputs, noisy inputs, and visible inputs. We use solid and dashed lines to represent the visible and hidden lines, respectively. In noisy line drawings (the second row), we use blue lines to represent noisy lines and highlight the missing lines using the red circle.

1. Introduction

In this paper, we tackle a long-standing problem in computer-aided design (CAD), namely 3D object reconstruction from three orthographic views. In today's product design and manufacturing industry, 2D engineering drawings are commonly used by designers to realize, update, and share their ideas, especially during the initial design stages. But to enable further analysis (*e.g.*, finite element analysis) and manufacturing, these 2D designs must be manually realized as 3D models in CAD software. Therefore, if a method can automatically convert the 2D drawings into 3D models, it would greatly facilitate the design process and improve overall efficiency.

As the most popular way to describe an object in 2D drawings, an orthographic view is the projection of the object onto the plane that is perpendicular to one of the three principal axes (Figure 1). Over the past few decades, 3D reconstruction from three orthographic views has been extensively studied, with significant improvements in terms of the types of applicable objects and computational efficiency [25, 10, 19, 37, 38, 28, 18, 21, 8, 9]. However, to the best of our knowledge, these techniques have not enjoyed wide adoption in CAD software and commercial products.

Among the challenges faced by existing methods in practice, their sensitivity to errors and missing components in the drawings is arguably the most critical one. To understand this issue, we note that almost all existing methods follow a standard procedure for 3D reconstruction, which consists of the following steps: **(i)** generate 3D vertices from 2D vertices; **(ii)** generate 3D edges from 3D vertices;

†Work done during internship at Manycore Tech Inc.
*Equal contributions.

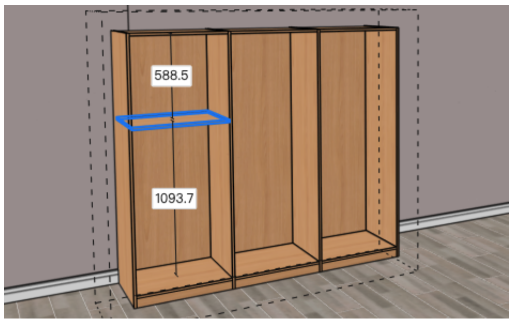

Figure 2. An illustration of cabinet design in a 3D modeling software.

(iii) generate 3D faces from 3D edges; and **(iv)** construct 3D models from 3D faces (see Figure 6 for an illustration). One main benefit of following the pipeline is that all solutions that match the input views can be found, as it establishes explicit correspondences between entities in the 3D model and those in the drawing. But in practice, rather than making an extra effort to perfect the drawings, designers would deem a drawing good enough as long as it conveys their ideas. Hence, some entities may be erroneous or missing. As a result, the aforementioned pipeline often fails to find the desired solution.

To overcome this difficulty, therefore, it is necessary to reason about the 2D drawings in a more holistic manner, and enable more flexible mappings between the input and output. Recently, Transformer [29] has become the standard architecture in many NLP and CV tasks. It is particularly effective in sequence-to-sequence (seq2seq) problems, such as machine translation, where reasoning about the context and soft alignment between the input and output are critical. Motivated by this, we convert our problem into a seq2seq problem and propose a Transformer-based deep learning method. Intuitively, the self-attention modules allow the model to capture the intent of the product designers even if their drawings are imperfect, and the cross-attention modules enable flexible mappings between geometric entities in the 2D drawing and 3D model.

Another benefit of employing learned representations and soft alignments for geometric entities is that one is free to choose how the 3D model is constructed. This provides us with opportunities to incorporate domain knowledge in our method to boost its performance. To illustrate this, we focus on a specific type of product, *cabinet furniture*, in this paper. As illustrated in Figure 2, a cabinet is typically built by arranging and attaching a number of planks (*i.e.*, wooden boards) together in a 3D modeling software. To this end, we develop a simple domain-specific language (DSL) based around declaring planks and then attaching them to one another, so that each cabinet can be represented by a program. Finally, given the input orthographic views, we train the Transformer-based model to predict the program associated with the cabinet.

To systematically evaluate the methods, we build a new benchmark dataset consisting of more than 26,000 3D cabinet models for this task. Most of them are created by professional interior designers using commercial 3D modeling software. Extensive experiments show that our method is much more robust to imperfect inputs. For example, the traditional method achieves an F1 score of 8.20% when 30% of the lines are corrupted or missing in the input drawings, whereas our method achieves an F1 score of 90.14%.

In summary, the contributions of this work are: **(i)** To the best of our knowledge, we are the first to use deep generative models in the task of 3D CAD model reconstruction from three orthographic views. Compared to existing methods, our model learns a more flexible mapping between the input and output, thus being more robust to noisy or incomplete inputs. **(ii)** We propose a new network design that learns shape programs to assemble planks into 3D cabinet models. Such a design not only improves reconstruction accuracy but also facilitates downstream applications such as CAD model editing.

2. Related Work

3D reconstruction from three orthographic views. Studies on recovering 3D models from three orthographic views date back to the 70s and 80s [13, 22, 32, 25, 10]. An early survey on this topic appears in [31]. According to [31], to obtain 3D objects in the boundary representation (B-rep) format, existing methods follow a four-stage scheme in which 3D vertices, edges, faces, and blocks are gradually built upon the results of previous steps. As mentioned before, a key strength of the framework is that all possible solutions that exactly match the input views can be found.

Subsequent methods for this task [37, 38, 28, 18, 21, 8, 9] also follow the same procedure and focus on extending the methods' applicable domain to cover more types of objects. For example, Shin and Shin [28] developed a method to reconstruct objects composed of planar and limited quadric faces, such as cylinders and tori, that are parallel to one of the principal axes. To remove the restriction placed on the axes of curved surfaces, Liu *et al.* [21] designed an algorithm that combines the geometric properties of conics with affine properties. Later, Gong *et al.* [9] proposed to recognize quadric surface features via hint-based pattern matching in the Link-Relation Graph (LRG), expanding the applicable domain to cover objects like those with interacting quadric surfaces. However, all these methods assume clean inputs and, as we will show in the experiment section, could easily break down in the presence of errors and noises.

Recently, Han *et al.* [12] also trained a deep network to reconstruct a 3D model from three orthographic views. However, their method takes raster images as input and produces results in the format of unstructured point clouds,

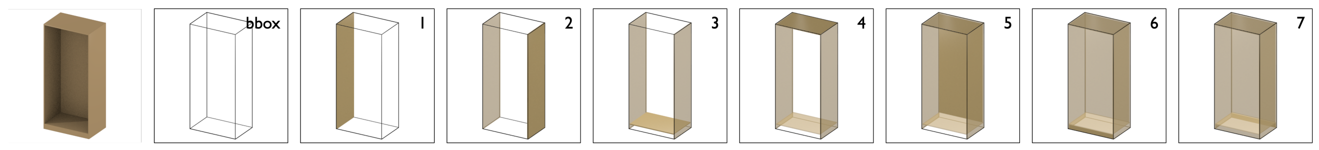

Figure 3. An illustration of how a simple cabinet is incrementally constructed by executing the shape program commands. The corresponding shape program is shown in Program 1.

which are of little use in CAD modeling applications. In contrast, our method directly uses vectorized line drawings as input and generates structured CAD models as output.

Deep generative models for CAD. With the availability of large-scale CAD datasets such as ABC [17] and Fusion 360 Gallery [34], a line of recent work trains deep networks to generate structured CAD data in the form of 2D sketches [33, 7, 27] or 3D models [35, 14, 11, 36]. These methods all cast it as a sequence generation problem, but differ in the DSLs used to produce the output. Our idea to generate cabinet furniture by assembling plank models together is inspired by ShapeAssembly [15], which learns to generate objects as hierarchical 3D part graphs. However, unlike the above studies, which focus on the generative models themselves, we propose to use generative models to build effective and efficient method for 3D CAD model reconstruction from three orthographic views.

3. A Simple Assembly Language for Cabinets

In this section, our goal is to define a domain-specific language (DSL) for the shapes of interest (*i.e.*, cabinet furniture). With this language, each cabinet model can be represented by a shape program, which will later be converted into a sequence of tokens as the output of a Transformer-based seq2seq model.

We define our DSL in the way that the resulting shape program resembles how a human designer builds the model in 3D modeling software. As shown in Figure 2, a cabinet is typically assembled by a list of plank models. In practice, most planks are axis-aligned cuboids. Therefore, we use cuboid as the only data type in our language. In Section 6, we discuss how our approach may be extended to accommodate more complex shapes (*e.g.*, a plank with a non-rectangular profile).

An axis-aligned cuboid has six degrees of freedom (DOF), which correspond to the starting and ending coordinates along the three axes:

$$\texttt{Cuboid}(x_{\min}, y_{\min}, z_{\min}, x_{\max}, y_{\max}, z_{\max}). \quad (1)$$

In practice, instead of specifying the numerical values for all the coordinates, human designers frequently use the *attachment* operation. As a form of geometric constraints, the benefit of using attachment is at least two-fold: *First*, it enables users to quickly specify the location of a plank without explicitly calculating (some of) its coordinates; *Second*,

```
bbox = Cuboid(-0.35, -0.23, -0.76, 0.35, 0.23, 0.76)
plank1 = Cuboid(bbox_1, bbox_2, bbox_3, -0.34, bbox_5, bbox_6)
plank2 = Cuboid(0.34, bbox_2, bbox_3, bbox_4, bbox_5, bbox_6)
plank3 = Cuboid(plank1_4, bbox_2, -0.70, plank2_1, bbox_5,
      -0.69)
plank4 = Cuboid(plank1_4, bbox_2, 0.75, plank2_1, bbox_5,
      bbox_6)
plank5 = Cuboid(plank1_4, 0.21, plank3_6, plank2_1, 0.22,
      plank4_3)
plank6 = Cuboid(plank1_4, bbox_2, bbox_3, plank2_1, -0.21,
      plank3_3)
plank7 = Cuboid(plank1_4, 0.21, bbox_3, plank2_1, bbox_5,
      plank3_3)
```

Program 1. The subscript indicates the index of coordinate values.

it facilitates future edits as any changes made to a plank will be automatically propagated to the others. Take Figure 2 as an example. When adding a plank (highlighted in blue), a designer may attach its four sidefaces to existing planks (including the invisible bounding box), while specifying the distances to the top and bottom in numerical values.

```
union Coord{
  float v;
  float* p;
};
```

Our language supports specifying the plank coordinates via either numerical values or attachment operation by adopting the `Union` structure commonly used in programming languages (*e.g.*, C++). As shown in the figure to the right, each of the six coordinates in Eq. (1) can either take a numerical value or be a pointer to the corresponding coordinate of another cuboid (to which it attaches to). Figure 3 shows an example cabinet incrementally constructed by imperatively executing the program commands (Program 1).

Shape program as a DAG. Alternatively, we may interpret the shape program as a directed acyclic graph (DAG). Note that each plank model consists of six faces, where each face corresponds to exactly one DOF in the axis-aligned cuboid (*i.e.*, $x_{\min}, y_{\min}, z_{\min}, x_{\max}, y_{\max}, z_{\max}$). Therefore, each program can be characterized by a graph $\mathcal{G} = \{\mathcal{F}, \mathcal{E}\}$, whose vertices $\mathcal{F} = \{f_1, \dots, f_{|\mathcal{F}|}\}$ represent the faces of plank models and whose edges $\mathcal{E} = \{e_1, \dots, e_{|\mathcal{E}|}\}$ represent attachment relationships between faces. Each directed edge $e_{i \to j}$ is an ordered pair of vertices (f_i, f_j), indicating the i-th face f_i attaches to the j-th face f_j. We assume that each face can attach to at most one another face; that is, the out-degree of any face f_i is at most one. Further, the edges $\mathcal{E}$ can be represented by an adjacency matrix $A \in \mathbb{R}^{|\mathcal{F}| \times |\mathcal{F}|}$. Specifically, A_{ij} is 1 if f_i directs to f_j, and 0 otherwise.

4. The PlankAssembly Model

As shown in Figure 1, we assume the input consists of three orthographic projections of the object, namely, the front view, top view, and side view: $\mathcal{V} = \{V_F, V_T, V_S\}$. Each view can be regarded as a planar graph of 2D edges and node points where the edges meet. We use solid lines to represent visible edges and dashed lines to represent hidden edges. Our goal is to reconstruct a 3D cabinet model described by the shape program or the equivalent DAG $\mathcal{G}$.

In this paper, we cast 3D reconstruction as a seq2seq problem. In Sections 4.1 and 4.2, we describe how to encode the input views $\mathcal{V}$ and the shape program $\mathcal{G}$ as 1D sequences $\mathcal{V}^{\text{seq}}$ and $\mathcal{G}^{\text{seq}}$, respectively. Then, we introduce the design of our PlankAssembly model, which adopts a Transformer-based encoder-decoder architecture to learn the probability distribution $p(\mathcal{G}^{\text{seq}} \mid \mathcal{V}^{\text{seq}})$, in Section 4.3. Finally, we present implementation details in Section 4.4.

4.1. Input Sequences and Embeddings

For the input conditions, we first order the 2D edges in $\mathcal{V}$ by the views. Each 2D edge is written as (x_1, y_1, x_2, y_2), where we order its two endpoints from lowest to highest by the x-coordinate, followed by the y-coordinate (if $x_1 = x_2$). Then, we order a set of 2D edges by x_1, followed by x_2, y_1, and y_2. Next, we flatten all the edges into a 1D sequence $\mathcal{V}^{\text{seq}} = \{v_1, \dots, v_{N_v}\}$. Note that, since each 2D edge has four DOFs (*i.e.*, the x- and y-coordinates of two endpoints), the length of $\mathcal{V}^{\text{seq}}$ is $N_v = 4N_{\text{edge}}$, where N_{edge} is the total number of 2D edges in all three orthographic views.

We embed the i-th token v_i as:

$$E(v_i) = E_{\text{value}}(v_i) + E_{\text{view}}(v_i) + E_{\text{edge}}(v_i) + E_{\text{coord}}(v_i) + E_{\text{type}}(v_i), \quad (2)$$

where the value embedding E_{value} indicates the quantized coordinate value of the token, the view embedding E_{view} indicates which view (*i.e.*, the front, top, or side view) the 2D edge is from, the edge embedding E_{edge} indicates the relative position of the 2D edge in the corresponding view, and the coordinate embedding E_{coord} indicates the relative position of the coordinate in the corresponding 2D edge. Finally, we use a type embedding E_{type} to indicate whether the 2D edge is visible or hidden. In this paper, we quantize the coordinate values into 9-bit integers and use learned 512-D embeddings for each term in Eq. (2).

4.2. Output Sequences and Embeddings

To generate the shape program sequentially, we need to map the graph $\mathcal{G}$ to a sequence $\mathcal{G}^{\text{seq}}$. This requires us to define a vertex order π on $\mathcal{G}$: We first sort the vertices topologically, ensuring that direct successors are listed before their corresponding direct predecessors. Then, vertices that are not directly connected are ordered by the coordinate values. This gives us a sorted graph $\mathcal{G}^\pi$ whose vertices $\mathcal{F}^\pi$ follow the order π.

Since we would like to capture the modeling process and facilitate future editing, we prioritize attachment relationships over geometric entities. Similar to the input sequence encoding, we flatten $\mathcal{G}^\pi$ to obtain a 1D sequence $\mathcal{G}^{\text{seq}}$. The i-th element of the sequence $\mathcal{G}^{\text{seq}}$ can be obtained as:

$$g_i = \begin{cases} f_i^\pi, & \text{if } A_{ij}^\pi = 0, \forall j, \\ e_{i\to j}^\pi, & \text{if } A_{ij}^\pi = 1. \end{cases} \quad (3)$$

Further, we use two special tokens, `[SOS]` and `[EOS]`, to indicate the start and end of the output sequence, respectively.

For the inputs to the decoder of our model, we embed the token g_i using the associated face f_i^π as follows:

$$E(g_i) = E(f_i^\pi) = E_{\text{value}}(f_i^\pi) + E_{\text{plank}}(f_i^\pi) + E_{\text{face}}(f_i^\pi). \quad (4)$$

The value embedding E_{value} indicates the quantized coordinate value, which is shared for the input and output sequences. The plank embedding E_{plank} indicates the location of the corresponding plank in the cabinet model, and the face embedding E_{face} indicates the relative position of the face within the plank.

If the token corresponds to an edge $e_{i\to j}^\pi$ in $\mathcal{G}$, we identify the face f_j^π to which the current face f_i^π attaches, and use the same value embedding as f_j^π.

4.3. Model Design

To tackle this seq2seq problem, we factorize the joint distribution over the output sequence into a series of conditional distributions:

$$p(\mathcal{G}^{\text{seq}} \mid \mathcal{V}^{\text{seq}}) = \prod_t p\left(g_t \mid \mathcal{G}^{\text{seq}}_{<t}, \mathcal{V}^{\text{seq}}\right). \quad (5)$$

Here, since g_t may take the form of either a geometric entity (*i.e.*, f_i^π) or an attachment relationship (*i.e.*, $e_{i\to j}^\pi$), we need to generate a probability distribution over a fixed-length vocabulary set (of quantized coordinate values) *plus* a variable-length set of tokens $\mathcal{G}^{\text{seq}}_{<t}$ in the output sequence.

The former distribution is a categorical distribution, which is commonly used in classification tasks. Let $\mathbf{h}_t$ be the hidden feature obtained by the decoder at time t, we project it to the size of the vocabulary via a linear layer, which is then normalized to form a valid distribution:

$$p_{\text{vocab}}(g_t \mid \mathcal{G}^{\text{seq}}_{<t}, \mathcal{V}^{\text{seq}}) = \text{softmax}\left(\text{linear}\left(\mathbf{h}_t\right)\right). \quad (6)$$

To generate a distribution over the output sequence $\mathcal{G}^{\text{seq}}_{<t}$ at time t, we adopt the Pointer Networks [30]. Specifically, we first use a linear layer to predict a pointer. The pointer is then compared with the hidden features of all former steps

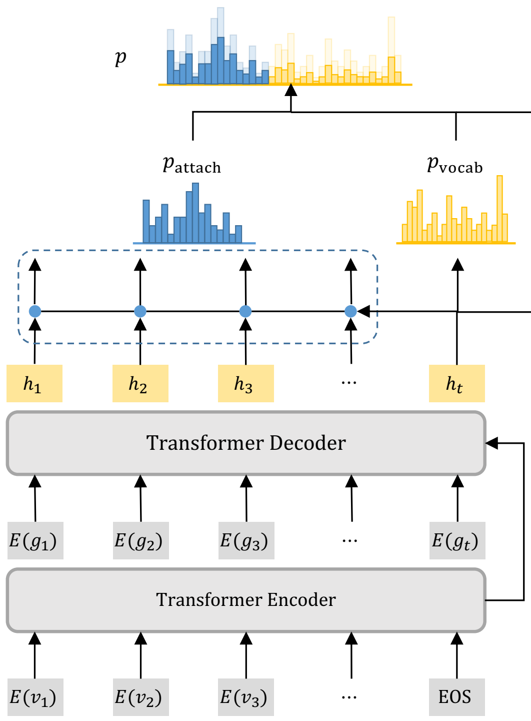

Figure 4. Network architecture. Our model takes the line sequences as input and outputs the shape program sequence autoregressively. At each time step t, the Transformer decoder outputs an attachment distribution p_{attach} over the previously predicted outputs, a vocabulary distribution p_{vocab}, and an attachment probability w_t. The final distribution p is obtained by concatenation of the two weighted distributions.

via dot-product. Finally, a distribution over the output sequence is obtained via a softmax layer:

$$p_{\text{attach}}(g_t \rightarrow g_k \mid \mathcal{G}^{\text{seq}}_{<t}, \mathcal{V}^{\text{seq}}) = \text{softmax}_k \left(\text{linear}\left(\mathbf{h}_t\right)^T \mathbf{h}_{<t}\right). \quad (7)$$

Instead of directly comparing these two distributions, we follow Pointer-Generator Networks [26] and introduce an attachment probability w_t to weight these two distributions. The attachment probability w_t is obtained via a linear layer and a sigmoid function $\sigma(\cdot)$: $w_t = \sigma\left(\text{linear}\left(\mathbf{h}_t\right)\right)$. Thus, the final distribution is the concatenation of the two weighted distributions:

$$p(g_t \mid \mathcal{G}^{\text{seq}}_{<t}, \mathcal{V}^{\text{seq}}) = \text{concat}\left\{(1 - w_t) \cdot p_{\text{vocab}}, w_t \cdot p_{\text{attach}}\right\}. \quad (8)$$

Finally, given a training set, the parameters of the model can be learned by maximizing the conditional distributions Eq. (8) via a standard cross-entropy loss.

Network architecture. We use the standard Transformer blocks [29] as the basic blocks of our PlankAssembly model. Given the input embeddings $\{E(v_1), E(v_2), \ldots\}$, the encoder encodes them into contextual embeddings. At decoding time t, the decoder produces hidden feature $\mathbf{h}_t$ based on the contextual embeddings and the decoder inputs $\{E(g_1), E(g_2), \ldots\}$. We use 6 Transformer layers for both the encoder and the decoder. Each layer has a feed-forward dimension of 1024 and 8 attention heads. The network architecture is summarized in Figure 4.

4.4. Implementation Details

Training. We implement our models with PyTorch Lightning [1]. We use 6 Transformer layers for both the encoder and the decoder. Each layer has a feed-forward dimension of 1024 and 8 attention heads. The network is trained for 400K iterations on four NVIDIA RTX 3090 GPU devices. We use Adam optimizer [16] with a learning rate of 10^{-4}. The batch size is set to 16 per GPU.

Inference. At inference time, we take several steps to ensure valid predictions from our model. First, we observed that two attaching faces must correspond to the opposite DOFs on the same axis, in order to avoid any spatial conflicts. For example, the x_{min} token of one plank can only point to the x_{max} token of another plank, and vice versa. Thus, we mask all invalid positions during inference. Second, we filter out the predicted planks with zero volume.

5. Experiments

5.1. Experimental Setup

Dataset. We create a large-scale benchmark dataset for this task, taking advantage of access to a large repository of cabinet furniture models from Kujiale[1], an online 3D modeling platform in the interior design industry. Most models in the repository are created by professional designers using commercial parametric modeling software, and are used for real-world production.

Several rules are used to filter the data: (i) We remove duplicated 3D models based on the similarity of the three orthographic views; (ii) We exclude models with fewer than four planks, more than 20 planks, or more than 300 edges in total. The remaining data is randomly split into three parts: 24039 for training, 1329 for validation, and 1339 for testing. To synthesize the three orthographic views, we use the `HLRBRep_Algo` API from pythonOCC [3], which is built upon the Open CASCADE Technology modeling kernel [2].

For our task, we need to parse each parametric cabinet model into a shape program. We first obtain the planks by extracting the geometric entities in the cabinet model. Note that in the parametric modeling software, a plank is typically created by first drawing a 2D profile and then applying

[1] http://kujiale.com

the extrusion command. Thus, we categorize the faces of each plank into *sideface* or *endface*, depending on whether they are along the direction of the extrusion or not. Then, given a pair of faces from two different planks, we consider that an attachment relationship exists if (i) the two faces are within a distance threshold of 1mm, and (ii) the pair consists of one sideface and one endface. Finally, a directed edge from the endface to the sideface is added in $\mathcal{G}$.

Evaluation metrics. To evaluate the quality of the 3D reconstruction results, we use three standard metrics: precision, recall, and F1 score. Specifically, for a cabinet model, we use Hungarian matching to match the predicted planks and the ground truth planks. A prediction is considered a true positive if its 3D intersection-over-union (IOU) with one ground truth is greater than 0.5.

5.2. Comparison to Traditional Methods

In this section, we systematically compare our approach with the traditional methods for 3D reconstruction from three orthographic views. Since no implementation of the traditional pipeline is publicly available, we reimplement the pipeline by closely following prior work [25, 28]. Recall that, starting from the input views, the traditional pipeline generates 3D vertices, 3D edges, 3D faces, and 3D blocks step by step. Then, solutions are found by enumerating all combinations of the candidate blocks and checking if their 2D projections match the input views.

To make the pipeline suitable for reconstructing assembly models like cabinets, we introduce two minor adjustments to it. *First*, in the traditional pipeline, two blocks that share a common face are merged together, which leads to mismatching between projections and input line drawings. In our implementation, we simply omit the original merging operation. *Second*, the blocks generated by the traditional pipeline correspond to the minimal closed spaces in 3D. During the evaluation, they cannot be directly matched with ground truth planks through bipartite matching. Instead, we group the blocks as a single prediction if they overlap with the same ground-truth plank model. Note that the proposed adjustments slightly favor the traditional approach, as we use ground truth information to merge the blocks.

Moreover, in cases where the traditional approach generates multiple solutions that all satisfy the inputs, we randomly select one as the final output.

Experiment on varying input noise levels. We first study the performance of both methods on imperfect inputs. In this experiment, we inject varying levels of noise into the input views. We consider two types of noises/errors commonly seen in real-world drawings: missing lines and inaccurate endpoints. Specially, we randomly select a percentage of 2D edges in the input views. For each selected edge, we either delete it or randomly perturb its endpoints along

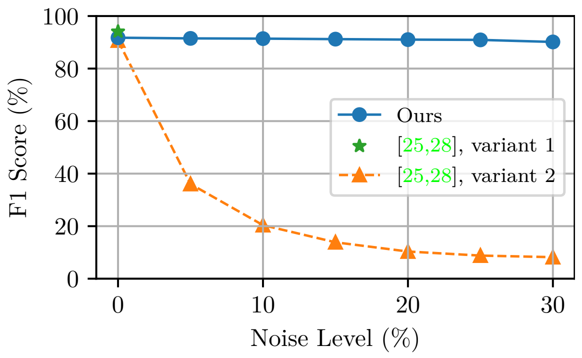

Figure 5. Comparison on varying input noise levels.

the edge direction. Figure 5 reports the F1 scores of both methods as the percentage varies from 0% to 30%.

Note that, when applied to our dataset, a major bottleneck of the traditional pipeline is the re-projection verification step in which all possible combinations of candidate blocks are projected to 2D to check if they match the input views. The reason is two-fold. *First*, such a match typically does not exist for noisy inputs. This is illustrated in Figure 6. When the input views contain errors, the 3D blocks generated by the traditional pipeline are often incomplete. Consequently, none of the combinations would exactly match the input views, resulting in a failure in the final solid reconstruction step. *Second*, even on clean inputs, the verification may take a long time due to a large number of possible combinations of the candidate blocks.

Thus, for completeness, we compare two variants of traditional pipeline in Figure 5. In *the first variant*, we enforce the re-projection verification step during reconstruction. On clean inputs, this variant achieves a slightly higher F1 score (94.07%) than ours (91.75%). However, this variant fails to produce a solution for 518 cases (out of 1339 test cases) in a reasonable time (5 minutes) due to exponential search complexity. Furthermore, this variant is not applicable to noisy inputs.

In *the second variant*, we ignore the re-projection verification step and directly use the union of blocks as the solution. As shown in Figure 5, it achieves an F1 score of 90.67% on clean input. And its performance degrades quickly as the input noise level increases. On the 30% noise level, this variant only produces results on 54 objects and has an F1 score of 8.20%.

In contrast, our approach is much more robust to input noises. Specifically, its performance only slightly drops from 91.75% to 90.14% as the noise level increases from 0% to 30%, verifying the key advantage of our method over traditional ones. In terms of inference time, our method takes about 0.63 seconds per sample on a single RTX 3090 GPU device.

Experiment on inputs with visible edges only. In real-

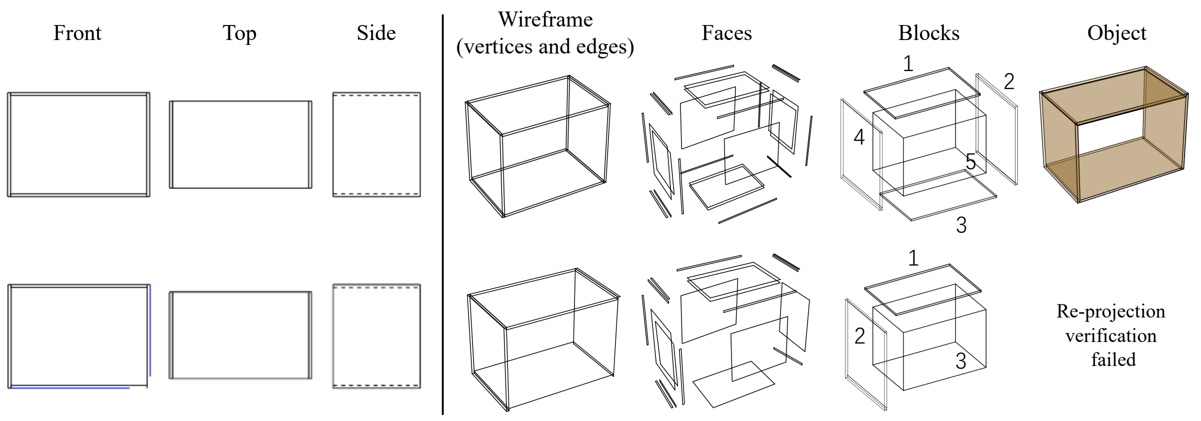
Front
Top
Side
Wireframe (vertices and edges)
Faces
Blocks
Object
1
2
3
4
5
1
2
3
Re-projection verification failed

Figure 6. A step-by-step illustration of the traditional approach on clean input **(top)** and noisy input **(bottom)**. The traditional pipeline accurately reconstructs the object when the input line drawings are free of noise. However, it fails to recover all the 3D blocks in the presence of noises, resulting in a failure of the re-projection verification in the final solid reconstruction step.

Methods	Precision	Recall	F1 score
[25, 28]	**99.64**	26.47	39.31
Ours	84.12	**82.05**	**82.62**

Table 1. Comparison on inputs with visible edges only.

world design practice, it is common for designers to omit the hidden edges of line drawings. Although it is still easy for humans to infer the 3D model, the traditional approach is likely to fail since the inputs are highly incomplete. To further demonstrate the robustness of our method, we conduct an experiment in which only the visible parts of the line drawings as used as input. For this experiment, we remove all invisible edges in the training set and follow the same protocol in Section 4.4 to train our network from scratch.

As shown in Table 1, the traditional approach performs poorly on this task, with very low recall and F1 score. This is expected because, on average, invisible edges account for about 48% of the edges in a line drawing. Meanwhile, our method is robust to the incomplete inputs, achieving an F1 score of 82.62%.

Qualitative results. Figure 7 visualizes some 3D reconstruction results of the two methods. In the *first and second rows*, we show results with clean inputs. Our method correctly reconstructs all four objects, whereas the traditional pipeline fails on the last two objects. Specifically, for the first object in the second row, the traditional pipeline produces multiple solutions, and an incorrect one is selected as the final output. And for the second object, it fails to produce any result within the time budget (5 minutes).

In the *third to fifth rows*, we show results from inputs with noise levels 10%, 20%, and 30%, respectively. As one can see, the performance of traditional pipeline degrades quickly. In particular, it fails to find any valid blocks for the two cases with noise level 30%. In contrast, our method correctly reconstructs all six objects in the three rows.

Finally, in the *sixth row*, we show two cases from inputs with visible edges only. Again, the traditional approach performs poorly in these cases, whereas our method correctly recovers both objects.

5.3. Ablation Studies

Next, we investigate the effect of several design choices we made in the PlankAssembly model.

Ablation study on the input. First, we study the performance of our model with different types of inputs. In PlankAssembly, we directly use the sequence of 2D edges as input. Here, we consider two alternatives:

Image: Many deep networks for 3D reconstruction use raster images as input. Inspired by Atlas [23], we replace the Transformer encoder in PlankAssembly with a CNN-based feature extractor to construct a 3D feature volume from posed images. Specifically, we use ResNet50-FPN [20] to extract 2D features from each view. Then, we aggregate the 2D features into a 3D feature volume with known poses and use a 3D CNN to refine the 3D features. The Transformer decoder takes the flattened features as input and outputs the shape program.

Sideface: Given the vectorized 2D line drawings, it is also possible to extract 2D sidefaces (*i.e.*, rectangles that correspond to the 3D sidefaces of the plank models) and use the sequence of sidefaces as input. Intuitively, this allows the seq2seq model to leverage explicit correspondences between the input and output. To this end, we design a set of heuristic rules to extract sidefaces in each view: First, we use the `polygonize` API from Shapely [4] to construct minimal closed polygons from each line drawing. Then, each sideface is represented by a polygon's axis-aligned bounding box (AABB). Here, we exclude AABBs whose short side is larger than ϵ (those AABBs typically correspond to the profile faces of the planks). We also merge

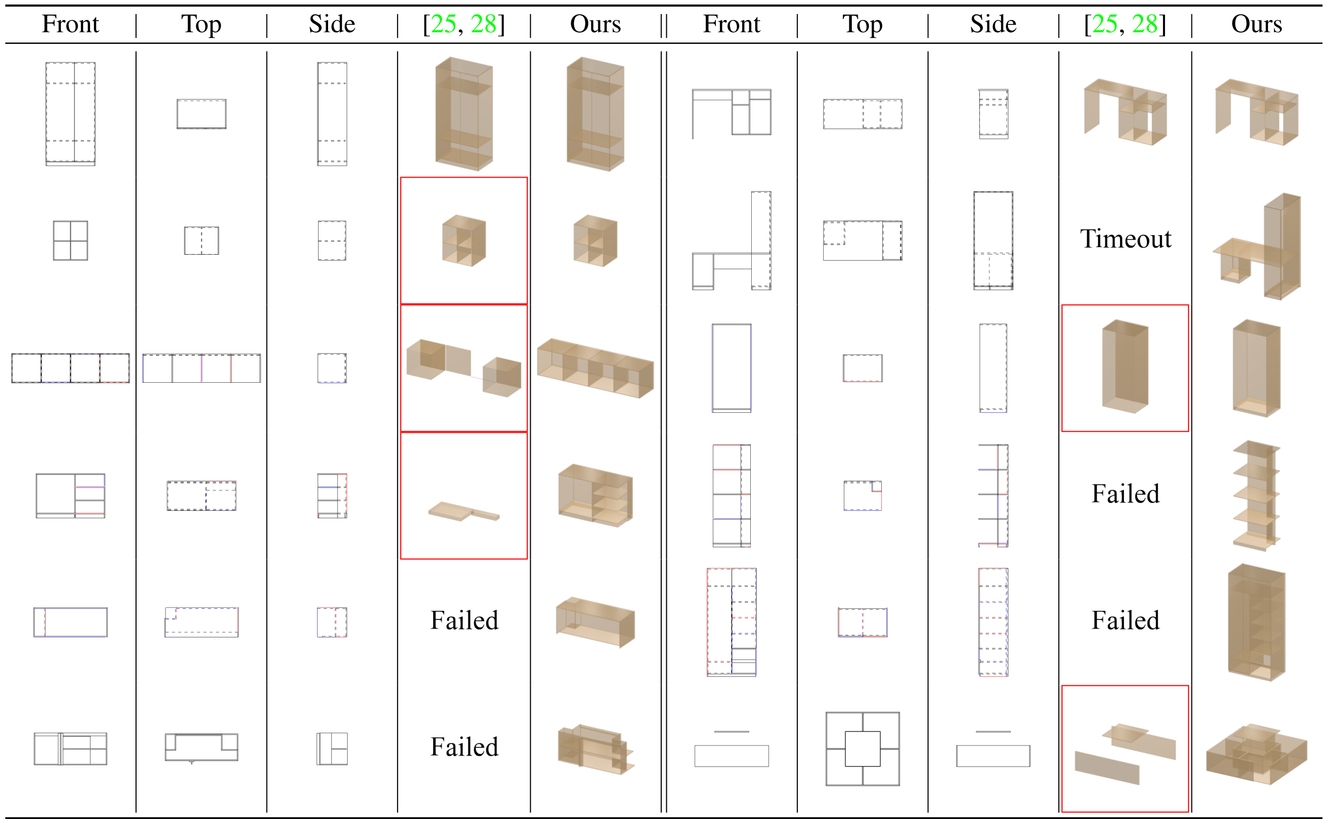

Figure 7. Qualitative results. **Rows 1-2**: clean inputs. **Rows 3-5**: Noisy inputs with noise level 10%, 20%, and 30%, respectively. **Row 6**: Inputs with visible parts only. We use red boxes to indicate incorrect reconstructions.

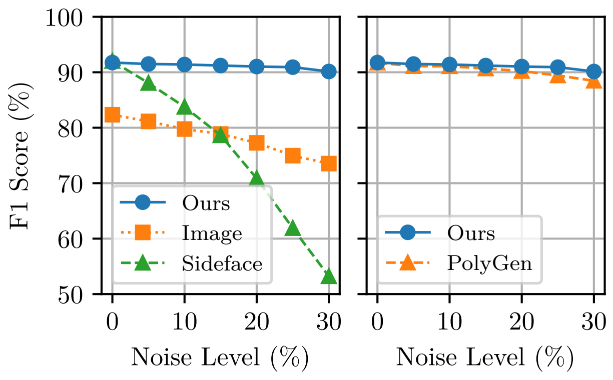

Figure 8. Ablation studies on the input sequence **(left)** and the output sequence **(right)**.

sidefaces recurrently if their short sides are adjacent. We set $\epsilon = 50$mm in our experiments.

Figure 8 (left) shows the performance of our method w.r.t. different types of inputs. As one can see, using raster images as inputs results in lower F1 scores across all noise levels, possibly because the features extracted from the line drawings are very sparse when treated as images. Further, the model achieves similar accuracies on clean inputs when using lines or sidefaces. However, the performance of the model using sidefaces is more sensitive to noises in the input views. This again shows that attempts to establish explicit correspondences between input and output hurt the methods' robustness – a phenomenon already seen in the comparison to traditional methods.

Ablation study on the output sequence. In PlankAssembly, we use shape programs as the outputs. In this experiment, we compare this choice to PolyGen [24], a popular approach to generate geometric models in the form of n-gon meshes. Similar to our method, PolyGen adopts a Transformer-based architecture and proceeds by generating a set of 3D vertices, which are then connected to form 3D faces. To obtain the planks in the cabinet models, we borrow the block generation step in the traditional pipeline to construct closed solids from the predicted faces.

The results are shown in Figure 8 (right). Our approach outperforms PolyGen, especially with high noise levels. Besides, our method runs about six times faster than PolyGen (0.63 *vs*. 3.61 seconds per sample), partly because PlankAssembly directly generates planks as the output and has a shorter output sequence (solids *vs*. vertices+faces).

Figure 9 compares the 3D models generated by our method and PolyGen. One notable issue with PolyGen is that since it generates each face separately, there is no guarantee that the faces will form closed solids (*i.e*., planks). For example, in the second row of Figure 9, one face predicted

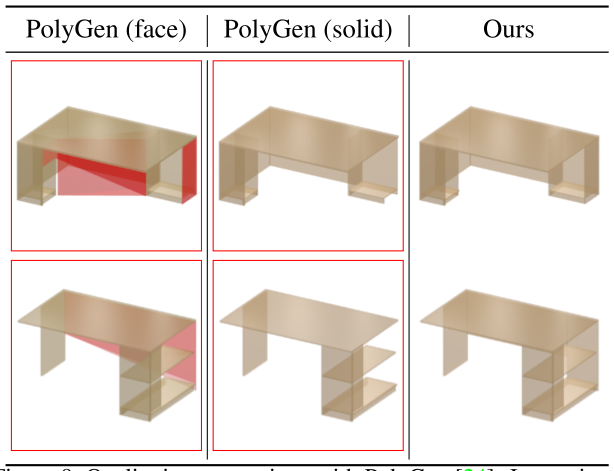

Figure 9. Qualitative comparison with PolyGen [24]. Input views are omitted. For PolyGen, we show results both before and after the solid construction step. Some incorrectly reconstructed faces are highlighted in red.

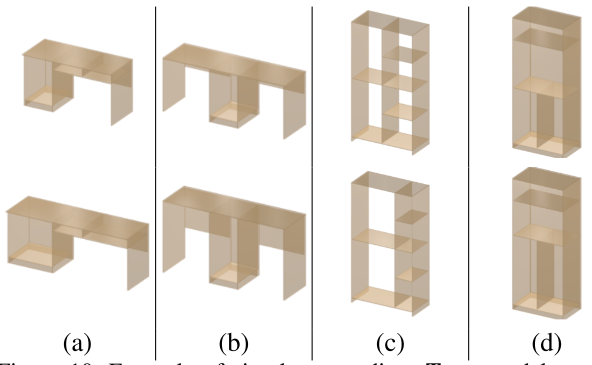

Figure 10. Example of simple user edits. **Top:** models reconstructed by PlankAssembly. **Bottom:** edited models.

by PolyGen has a non-rectangular shape, leading to missing planks after the solid construction step.

Another benefit of leveraging domain-specific language over general geometric forms such as n-gons is that the generated shapes can better support user edits in the CAD modeling software. Specifically, given the attachment relationships predicted by our method, a cabinet model may undergo global scaling operations (Figure 10 (a-b)) or local editing operations (Figure 10 (c-d)) while maintaining the correct topology.

5.4. Failure Cases

Figure 11 illustrates two most common failure modes of our PlankAssembly model. In the first example, the network makes incorrect predictions for attachments. In the second example, the reconstructed 3D model is incomplete because the stop token is predicted too early, which is a known issue with auto-regressive models.

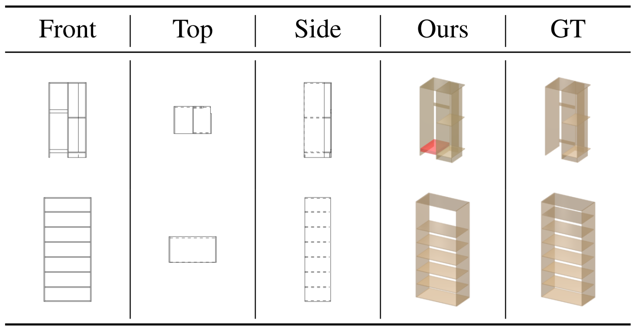

Figure 11. Failure cases. We highlight incorrectly reconstructed planks in red.

6. Discussion

This paper advocates a *generative approach* to 3D CAD model reconstruction from three orthographic views. Two lessons can be learned from our experiments: *First*, compared to finding explicit correspondences between the 2D line drawings and 3D models, the attention mechanism plays a key role in the deep network's robustness to the imperfect inputs. *Second*, incorporating domain knowledge in the generative model benefits both the reconstruction and downstream applications.

One may argue that our experiments are limited to cabinet furniture, a special type of CAD model. However, we emphasize that our main idea and the lessons learned are general and can be applied to any CAD model. For example, prior work such as DeepCAD [35] has developed neural networks which are able to generate CAD command sequences suitable for mechanical parts. Unlike cabinet furniture, mechanical parts often have non-rectangular profiles (but fewer blocks). It is thus relatively straightforward to extend our approach to such domains.

A more challenging scenario is one attempting to apply our data-driven approach to domains where large-scale CAD data is unavailable or even nonexistent, such as buildings or complex mechanical equipment. Besides, our current approach does not consider other information available in CAD drawings, such as layers, text, symbols, and annotations. Recently, several methods have been proposed for panoptic symbol spotting in CAD drawings [6, 39, 5]. We believe that such information is also vital for 3D reconstruction from complex CAD drawings.

Acknowledgements

This work was supported in part by the Key R&D Program of Zhejiang Province (2022C01025). Jian Yin is supported by the National Natural Science Foundation of China (U1911203, U2001211, U22B2060), Guangdong Basic and Applied Basic Research Foundation (2019B1515130001), Key-Area Research and Development Program of Guangdong Province (2020B0101100001).

References

[1] Lightning AI. https://github.com/Lightning-AI/lightning. 5

[2] Open CASCADE Technology. https://dev.opencascade.org/. 5

[3] pythonocc. https://github.com/tpaviot/pythonocc-core. 5

[4] shapely. https://github.com/shapely/shapely. 7

[5] Zhiwen Fan, Tianlong Chen, Peihao Wang, and Zhangyang Wang. Cadtransformer: Panoptic symbol spotting transformer for CAD drawings. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10976–10986, 2022. 9

[6] Zhiwen Fan, Lingjie Zhu, Honghua Li, Xiaohao Chen, Siyu Zhu, and Ping Tan. Floorplancad: A large-scale CAD drawing dataset for panoptic symbol spotting. In *Int. Conf. Comput. Vis.*, pages 10108–10117, 2021. 9

[7] Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. In *Adv. Neural Inform. Process. Syst.*, pages 5885–5897, 2021. 3

[8] Jie-Hui Gong, Gui-Fang Zhang, Hui Zhang, and Jia-Guang Sun. Reconstruction of 3d curvilinear wire-frame from three orthographic views. *Comput. Graph.*, 30(2):213–224, 2006. 1, 2

[9] Jie-Hui Gong, Hui Zhang, Gui-Fang Zhang, and Jia-Guang Sun. Solid reconstruction using recognition of quadric surfaces from orthographic views. *Comput. Aided Des.*, 38(8):821–835, 2006. 1, 2

[10] Kaining Gu, Zesheng Tang, and Jiaguang Sun. Reconstruction of 3d objects from orthographic projections. *Comput. Graph. Forum*, 5(4):317–323, 1986. 1, 2

[11] Haoxiang Guo, Shilin Liu, Hao Pan, Yang Liu, Xin Tong, and Baining Guo. ComplexGen: CAD reconstruction by b-rep chain complex generation. *ACM Trans. Graph.*, 41(4):129:1–129:18, 2022. 3

[12] Wenyu Han, Siyuan Xiang, Chenhui Liu, Ruoyu Wang, and Chen Feng. SPARE3D: A dataset for spatial reasoning on three-view line drawings. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 14678–14687, 2020. 2

[13] Masanori Idesawa. A system to generate a solid figure from three view. *Bulletin of the JSME*, 16(92):216–225, 1973. 2

[14] Pradeep Kumar Jayaraman, Joseph G. Lambourne, Nishkrit Desai, Karl D. D. Willis, Aditya Sanghi, and Nigel J. W. Morris. SolidGen: An autoregressive model for direct b-rep synthesis. *Trans. Mach. Learn. Res.*, 2023. 3

[15] R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. ShapeAssembly: learning to generate programs for 3d shape structure synthesis. *ACM Trans. Graph.*, 39(6):234:1–234:20, 2020. 3

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2015. 5

[17] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A big cad model dataset for geometric deep learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 9601–9611, 2019. 3

[18] Mu-Hsing Kuo. Reconstruction of quadric surface solids from three-view engineering drawings. *Comput. Aided Des.*, 30(7):517–527, 1998. 1, 2

[19] Rémi Lequette. Automatic construction of curvilinear solids from wireframe views. *Comput. Aided Des.*, 20(4):171–180, 1988. 1

[20] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2117–2125, 2017. 7

[21] Shi-Xia Liu, Shi-Min Hu, Yujian Chen, and Jia-Guang Sun. Reconstruction of curved solids from engineering drawings. *Comput. Aided Des.*, 33(14):1059–1072, 2001. 1, 2

[22] George Markowsky and Michael A. Wesley. Fleshing out wire frames. *IBM J. Res. Dev.*, 24(5):582–597, 1980. 2

[23] Zak Murez, Tarrence van As, James Bartolozzi, Ayan Sinha, Vijay Badrinarayanan, and Andrew Rabinovich. Atlas: End-to-end 3d scene reconstruction from posed images. In *Eur. Conf. Comput. Vis.*, pages 414–431, 2020. 7

[24] Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. PolyGen: An autoregressive generative model of 3d meshes. In *Int. Conf. Mach. Learn.*, pages 7220–7229, 2020. 8, 9

[25] Hiroshi Sakurai and David C. Gossard. Solid model input through orthographic views. In *ACM SIGGRAPH*, pages 243–252, 1983. 1, 2, 6, 7, 8

[26] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *ACL*, pages 1073–1083, 2017. 5

[27] Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P. Adams. Vitruvion: A generative model of parametric CAD sketches. In *Int. Conf. Learn. Represent.*, 2022. 3

[28] Byeong-Seok Shin and Yeong-Gil Shin. Fast 3d solid model reconstruction from orthographic views. *Comput. Aided Des.*, 30(1):63–76, 1998. 1, 2, 6, 7, 8

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inform. Process. Syst.*, pages 5998–6008, 2017. 2, 5

[30] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Adv. Neural Inform. Process. Syst.*, pages 2692–2700, 2015. 4

[31] Weidong Wang and Georges G. Grinstein. A survey of 3d solid reconstruction from 2d projection line drawings. *Comput. Graph. Forum*, 12(2):137–158, 1993. 2

[32] Michael A. Wesley and George Markowsky. Fleshing out projections. *IBM J. Res. Dev.*, 25(6):934–953, 1981. 2

[33] Karl D. D. Willis, Pradeep Kumar Jayaraman, Joseph G. Lambourne, Hang Chu, and Yewen Pu. Engineering sketch generation for computer-aided design. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, pages 2105–2114, 2021. 3

[34] Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: a dataset and environment for programmatic CAD construction from human

design sequences. *ACM Trans. Graph.*, 40(4):54:1–54:24, 2021. 3

[35] Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A deep generative network for computer-aided design models. *Int. Conf. Comput. Vis.*, pages 6772–6782, 2021. 3, 9

[36] Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. Skexgen: Autoregressive generation of CAD construction sequences with disentangled codebooks. In *Int. Conf. Mach. Learn.*, pages 24698–24724, 2022. 3

[37] Qing-Wen Yan, C. L. Philip Chen, and Zesheng Tang. Efficient algorithm for the reconstruction of 3d objects from orthographic projections. *Comput. Aided Des.*, 26(9):699–717, 1994. 1, 2

[38] Chun-Fong You and Shih-Shing Yang. Reconstruction of curvilinear manifold objects from orthographic views. *Comput. Graph.*, 20(2):275–293, 1996. 1, 2

[39] Zhaohua Zheng, Jianfang Li, Lingjie Zhu, Honghua Li, Frank Petzold, and Ping Tan. GAT-CADNet: Graph attention network for panoptic symbol spotting in cad drawings. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 11737–11746, 2022. 9