Neural Face Identification in a 2D Wireframe Projection of a Manifold Object

Kehan Wang*
University of California, Berkeley
wang.kehan@berkeley.edu

Jia Zheng Zihan Zhou
Manycore Tech Inc.
{jiajia, shuer}@qunhemail.com

Abstract

In computer-aided design (CAD) systems, 2D line drawings are commonly used to illustrate 3D object designs. To reconstruct the 3D models depicted by a single 2D line drawing, an important key is finding the edge loops in the line drawing which correspond to the actual faces of the 3D object. In this paper, we approach the classical problem of face identification from a novel data-driven point of view. We cast it as a sequence generation problem: starting from an arbitrary edge, we adopt a variant of the popular Transformer model to predict the edges associated with the same face in a natural order. This allows us to avoid searching the space of all possible edge loops with various hand-crafted rules and heuristics as most existing methods do, deal with challenging cases such as curved surfaces and nested edge loops, and leverage additional cues such as face types. We further discuss how possibly imperfect predictions can be used for 3D object reconstruction. The project page is at https://manycore-research.github.io/faceformer.

1. Introduction

In this paper, we revisit a classical problem in computer-aided design (CAD), namely the conversion of 2D line drawings into 3D objects. In a traditional CAD pipeline, design engineers commonly draw a 2D wireframe[1] of the desired object when creating their ideas and when communicating their ideas to others. Therefore, there is a strong need to develop algorithms which can convert 2D line drawings into 3D solid models for analysis, simulation, and manufacturing.

An important step of 3D reconstruction from 2D line drawings is face identification, that is, finding loops of the edges which correspond to faces of the 3D object (Figure 1). If the correct face configuration of an object can be obtained, the number of degrees of freedom in its reconstruction will be greatly reduced. Many studies have been performed to achieve this goal, and in certain cases, a satisfactory solution exists. For example, it is well known that there is a unique planar embedding when the object has genus 0 and the drawing is 3-connected, from which the set of faces can be determined [24]. However, for a general manifold object with complex geometry (*e.g.*, curved surfaces) and topology (*e.g.*, high genus), the task remains difficult.

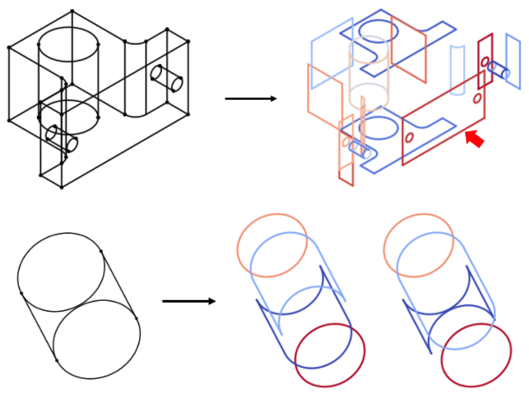
Figure 1. Given a 2D line drawing, face identification aims to find edge loops which correspond to actual faces of the 3D object. **First row:** A case where several faces (including the one indicated by a red arrow) are enclosed by two or more loops. **Second row:** A case where multiple topologically correct interpretations exist.

A close look at current methods reveals two primary sources of challenges. The first one is the algorithmic complexity. As no analytical solution is available for a general manifold object, these methods need to search through the space of all possible face loops, which grows exponentially with the number of vertices. In order to make the solution efficient, much effort has been made to design heuristic search algorithms [7, 18, 19, 27]. The second one is the inherent ambiguity in reconstructing 3D objects from 2D projections. For example, it is impossible for a topological algorithm to tell if two nested cycles (Figure 1, first row) are coplanar, thus are two loops of the same face. Besides, a single 2D projection may yield multiple topologically correct solutions (Figure 1, second row). In such cases, additional heuristics (*e.g.*, number of cycles, regularities) or

*Work done during an internship at Manycore Tech Inc.
[1]We use the terms "wireframe" and "line drawing" interchangeably.

human intervention is required to choose the final output.

Nevertheless, humans can effortlessly "see" the real faces in a typical line drawing, including those in Figure 1. Rather than performing a tedious search for the face loops, our ability to quickly identify the faces seems to be attributed to our past experience interacting with 3D objects. This makes us wonder: Is it possible for a computer to learn to recognize faces in a data-driven fashion? This work represents a first attempt to answer the question. We train a deep neural network to detect faces using a large collection of 3D objects and their 2D projections. This way, we are able to avoid exploring a search space of exponential order. Moreover, by learning from ground truth 3D data, our method implicitly learns to generate the most plausible solutions, resolving the inherent ambiguity associated with the problem.

To this end, we cast face identification as a sequence generation problem, leveraging the natural order of co-edges in a face loop (see Section 3 for details). Given a set of co-edges in a 2D line drawing, we train a variant of the popular Transformer model [28] to predict one co-edge index that forms the face at one timestamp. For each detected face, we further classify it into different types such as planes and cylindrical surfaces. On the public ABC dataset [15], our model achieves 93.8% and 95.9% in precision and recall, respectively. Finally, with the detected faces, we develop a simple convex optimization scheme to reconstruct structured 3D models from a single 2D line drawing.

In summary, the **main contributions** of this work are two-fold: (i) We propose a first data-driven approach to face identification and study its advantages and disadvantages over existing geometry- and topology-based methods. (ii) We develop a simple scheme to reconstruct a 3D model from a single 2D line drawing using the face identification results. With this work, we discuss new opportunities for incorporating learning-based approaches into established CAD pipelines, such as identifying conflicting face loops in a geometric constraint system for 3D modeling.

2. Related Work

Face identification and 3D reconstruction. Face identification is a long-standing problem in the area of automatic interpretation of line drawings. Research in this field dates back to the '70s and '80s. Early work [20] proposes a geometric approach to detecting planar faces in a 3D wireframe: it first generates possible planes at a vertex joined by two non-collinear edges, then searches for other vertices lying on each such plane and tries to use the vertices to form cycles. Subsequent work [1, 34] can deal with more general curve networks with highly curved faces and complex topology (*i.e.*, manifolds with high genus). But these methods rely on 3D coordinates of the wireframe, therefore cannot be applied to 2D projections.

Another line of work addresses the problem from a graph-topological point of view. To find the faces of a vertex-edge graph, Dutton and Brigham [6], Hanrahan [10] propose to compute the planar embedding of the graph, where the resulting regions represent the faces. Brewer and Courter [4], Ganter and Uicker [8] generate an initial cycle basis from the spanning tree of the graph, then perform a cycle reduction procedure to find the faces. But these methods are only suitable for genus-0 manifolds. For manifold objects of a non-zero genus or non-manifold objects, Shpitalni and Lipson [24] present a method which searches through the set of all possible sets of face loops and use various geometric criteria to assess them. The method could be slow as the number of possible face loops is $O(e^v)$ and any topological algorithm for finding faces of objects of genus > 0 has exponential complexity [2]. To reduce the search complexity, Liu et al. [18] propose a depth-first search that is primarily guided by topological constraints; Liu and Tang [19] develop a genetic algorithm which uses 2D geometric information in its assessment criteria; Varley and Company [27] use a heuristic search based on the shortest-path and Dijkstra's algorithm, and Fang et al. [7] introduce a fast algorithm via edge decomposition.

Given the face topology, several studies [17, 25, 30] reconstruct the 3D object from a single 2D line drawing. These works solve for the 3D shape in an optimization framework, using various constraints based on structural regularities, such as minimum standard deviation of angles (MSDA), face planarity, line parallelism, and corner orthogonality. In this work, we introduce a different method which uses the predicted face types – information previous methods do not have access to. Further, we study the impact of imperfect face detection results on 3D reconstruction.

Deep models. Our technical approach is inspired by the advance in sequence-to-sequence modeling [3, 28], which has produced the state-of-the-art results in a wide range of NLP and vision tasks lately. Specifically, our network design follows Pointer Net [29], which proposes an autoregressive model to generate a distribution on a given input data set. PolyGen [21] extends this idea to generate 3D polygon meshes by sequentially predicting the vertices and faces using a Transformer-based architecture.

Several recent papers also apply deep networks to CAD data [13, 16, 22, 31, 32, 33]. While our method shares common aspects with these work, such as network design [22, 31, 33] and the use of co-edges [16], we tackle a different problem in face identification in this paper.

3. Problem Formulation

In this paper, a 2D wireframe projection is assumed to be an orthographic projection where all the edges (including silhouettes) and vertices of the object are visible. We make a few assumptions about the input line drawing: *First*,

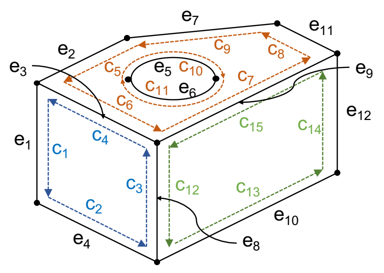

Figure 2. B-rep of an object. Hidden lines are omitted.

the hidden lines and vertices are given. *Second*, the crossing point of two edges in a line drawing is not a vertex and cannot be used to form faces. As such, the input line drawing can be represented as an edge-vertex graph $\mathcal{G} = (V, E)$, where each edge (or vertex) of the graph corresponds to exactly one edge (or vertex) of the object. Note that the graph may contain one or more connected components. In practice, these graphs may be a result of previous processing of a rough sketch or a scanned-in drawing [26].

Given the graph $\mathcal{G} = (V, E)$, the goal of face identification is to find all faces $F = \{f_1, \ldots, f_M\}$ of the object, where each face can be written as the set of enclosing edges: $f_i = \{e_{i_1}, \ldots, e_{i_n}\}$. Same as prior work [18, 27], we focus on *manifold objects* only. A manifold object is defined as a solid where every point on its surface has a neighborhood topologically equivalent to an open disk in the 2D Euclidean space. A key property of manifold objects is that each edge of a manifold is shared exactly by two faces. This property is best expressed in terms of *co-edges*, $C = \{c_1, c_2, \ldots\}$, an important type of topological entities in the B-rep. As illustrated in Figure 2, there is a co-edge pointing from vertex v_i to v_j if and only if there is an edge connecting v_i and v_j. For example, c_3 and c_{12} are two mutually mating co-edges associated with edge e_8.

A face can be conveniently represented as a loop (*i.e.*, closed path) of co-edges. For example, in Figure 2, the loops (c_1, c_2, c_3, c_4) and $(c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11})$ represent two faces of the object. We follow the conventional definition of loop direction: if a loop is viewed along its direction, with the face normal pointing upwards, then the face that owns the loop is to the left.

4. Face Identification via Sequence Generation

The natural order of co-edges corresponding to a face motivates us to treat face identification as a sequence generation problem. Specifically, a face with n co-edges can be written as: $f_i = (c_{i_1}, \ldots, c_{i_n})$, where each index i_t, $t = 1, 2, \ldots, n$, is an integer between 1 and N, and N is the total number of co-edges. Thus, starting with an arbitrary co-edge c_{i_1}, our goal is to grow it into a sequence of co-edges $(c_{i_1}, \ldots, c_{i_n})$ on which c_{i_1} lies, as shown in Figure 3. To detect all the faces $F = \{f_1, \ldots, f_M\}$, we may use every co-edge in C as the starting co-edge and repeat the process for N times.

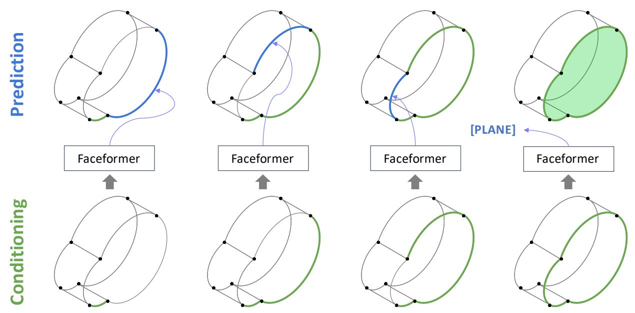

Figure 3. Our model, Faceformer, takes as input the set of all co-edges, and current sequence of co-edge indices, and outputs a distribution over the co-edge indices.

In the following, we describe how to generate a single face f_i given the starting co-edge c_{i_1}.

4.1. Face Identification Model

Since our goal is to select a subset of the input co-edges to form the output face, we first briefly review Pointer Net [29], a sequence-to-sequence (seq2seq) model which uses the attention mechanism to create pointers to input elements. Given the input P, the main idea is to learn the conditional probability $p(\mathcal{I} \mid P)$ using a parametric model, such as LSTM [12] or Transformer [28], to estimate the terms of the probability chain rule: $p(\mathcal{I} \mid P) = \prod_{t=1}^{T} p(i_t \mid i_1, \ldots, i_{t-1}, P)$. Here, $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots\}$ is a sequence of vectors and $\mathcal{I} = (i_1, \ldots, i_T)$ is a sequence of T indices, each between 1 and $|P|$.

Similar to most seq2seq models, Pointer Net adopts an encoder-decoder architecture. First, it obtains a contextual embedding $\mathbf{w}_k$ for each input using an encoder. At each decoding time step t, the decoder outputs a pointer vector $\mathbf{u}_t$, which is then compared to the contextual embeddings via a dot-product. The resulting scores are normalized using a softmax to form a valid distribution over the input set:

$$\{\mathbf{w}_k\}_{k=1}^{N} = \mathbf{Encoder}(P; \theta) \tag{1}$$

$$\mathbf{u}_t = \mathbf{Decoder}(\mathcal{I}_{<t}, P; \theta) \tag{2}$$

$$p(i_t = k \mid \mathcal{I}_{<t}, P; \theta) = \mathbf{softmax}_k(\mathbf{u}_t^T \mathbf{w}_k) \tag{3}$$

Finally, the parameters of the model are learned by maximizing the conditional probabilities on a training set.

Input sequence and embeddings. In our problem, the input sequence consists of all co-edges C. To further leverage geometric cues, we propose to classify the faces into different types – a benefit of data-driven approaches. In this work we only consider two special face types, namely planar surface and cylinder surface, but the method can be easily extended to other types. To this end, we add three special tokens, [PLANE], [CYLINDER] and [OTHERS], to indicate different face types. We replace regular stop tokens with the face type tokens, expecting the network to predict one of the three tokens immediately after generating all co-edges of the face.

We jointly embed the special tokens with the co-edges, to obtain a total of $N+3$ input embeddings. We use two embeddings for each input co-edge, including (i) value embedding, representing the coordinate value of the edge, and (ii) position embedding, indicating the token location in the sequence. Due to the varying edge length, we uniformly sample a fixed number of edge points to represent the co-edge as in [5]. The points are ordered based on the co-edge direction. Then, we flatten the edge points and apply two linear layers to obtain a 512-dimensional embedding.

Output sequence and embeddings. As mentioned above, we represent each face as a sequence of co-edges $f_i = (c_{i_1}, \dots, c_{i_n})$. A face type token is added (i) to predict the face type and (ii) to indicate the end of the sequence. When a face consists of multiple loops, special care needs to be taken to ensure a unique co-edge sequence. In such cases, except for the loop to which the starting co-edge belongs, we order the co-edges in each other loop from lowest to highest first by its x-coordinate, followed by y-coordinate. Take Figure 2 as an example, if we use c_{11} as the starting co-edge, the desired face sequence should be $(c_{11}, c_{10}, c_5, c_6, c_7, c_8, c_9)$.

Similar to input embeddings of the encoder, we use learned position and value embeddings for the inputs to the decoder. We use the co-edge's contextual embedding from the encoder output as its value embedding.

Network architecture. Inspired by recent success of Transformer-based architectures [28], we adopt it as the basic block of our face identification model. Given the input co-edge embeddings, the encoder encodes them into contextual embeddings, then the decoder outputs pointers based on the contextual embeddings and the decoder input. The model consists of 6 Transformer encoder and decoder layers, with a feed-forward dimension of 1024 and 8 attention heads.

4.2. Training and Inference

We implement our model with PyTorch and PyTorch Lightning. We use Adam optimizer [14] with a learning rate of 1×10^{-4}. The batch size is set to 4. The model is trained with 400 000 iterations, taking about 30 hours to converge on a single NVIDIA RTX 3090 GPU device.

At inference time, since our parallel model predicts faces from all co-edges independently, there are many duplicated faces in the result. We take three post-processing steps to remove invalid and duplicated predictions. First, because a face is defined as a closed path of co-edges, we filter out any unclosed face predictions. Second, we remove predictions that contain both co-edges associated with the same edge, as this never happens in real faces. Finally, we identify duplicated predictions by comparing the set of edges generated in each face prediction. Note that these duplicated predictions may have different face type classifications. We count the number of times each face type is predicted, and take the face type of the highest count as its predicted face type.

4.3. Experiments

4.3.1 Experimental Setup

Dataset. We build a benchmark for this novel task using a subset of CAD mechanical models from ABC dataset [15]. We use pythonOCC[2], a Python 3D development framework built upon the Open CASCADE Technology, to project CAD models into 2D line drawings. We first normalize the shape such that the half diagonal length of the bounding box is equal to 1. Then, the camera is placed at a random distance between 1.25 to 1.5 away from the object center, pointing towards the object. The viewpoints are randomly sampled on a hemisphere. The dataset consists of 9370/202/504 samples for training/validation/testing.

To eliminate cases that are unnecessarily complicated, we filter out shapes with more than 42 faces or 37 edges in a face. Since the ABC dataset has many duplicate shapes, we also run additional filters based on the shape's topology and three orthogonal views to remove duplicates.

Evaluation metrics. To evaluate the performance of face identification, we compute the *precision* and *recall* at the face level. We treat each face as an unordered set of edges. For our model, this is the result after the post-processing step as described in Section 4.2. A prediction f is said to match a ground truth face f^* if and only if the two sets are equal. Then, let F^* denote the set of ground truth faces, and F denote the set of faces detected by any method, the precision and recall are defined as: precision $= |F \bigcap F^*|/|F|$, recall $= |F \bigcap F^*|/|F^*|$.

For face type classification, we report the *classification rate*, which is the percentage of correctly classified faces among all faces correctly detected by our model.

4.3.2 Experimental Results

Comparison with prior work. We first compare our method with an existing method, FindingFaces [27]. To the best of our knowledge, this is the only related work with public code or implementation[3]. It assumes that, for each co-edge, the true face has the least cost (*e.g.*, the shortest path) among all loops enclosing that co-edge. Then, the Dijkstra's algorithm can be applied to find the least-cost closed

[2]https://github.com/tpaviot/pythonocc
[3]http://www.regeo.uji.es/FindingFaces.htm

method	precision	recall	runtime (s)
FindingFaces [27]	97.3	**97.8**	**0.008**
Ours	**98.2**	97.7	0.119

Table 1. Comparison with prior work (on the subset of polyhedral objects only). FindingFaces does not work on non-polyhedra.

models	precision	recall	runtime (s)
Seq2seq	77.6	76.8	0.69
Seq2seq + co-edge	**94.9**	90.5	0.65
Ours	93.8	**95.9**	**0.15**

Table 2. Experiment results on network design.

loops. But in practice, not all faces correspond to the shortest paths, so a more complicated "cost" is proposed in [27]. Besides, the order in which the co-edges are considered also affects its performance. Thus, much effort is made to develop heuristic rules for edge priorities.

Despite its efficiency, a key limitation of FindingFaces [27] is that it is applicable to polyhedral objects only. Among the 504 randomly sampled objects in our test set, only 126 (25%) are polyhedra. As shown in Table 1, both methods achieve high accuracies on this subset. Comparing to results on the full test set in Table 2, one can infer that polyhedra are relatively simple cases. Among the mistakes made by FindingFaces, a notable issue is that it cannot detect faces with more than one loop (*i.e.*, objects with holes) – a common problem for all topological approaches.

Experiment on network design. In this experiment, we compare our model with two variants of it to illustrate the benefits of (i) using co-edges and (ii) face-wise parallel prediction.

The first variant, *seq2seq*, directly operates on the edges (instead of co-edges) and generates all faces in a sequential fashion. In this variant, the input sequence would be the set of all edges, ordered from lowest to highest first by its x-coordinate, then by y-coordinate. For the output sequence, we introduce three additional special tokens, `[SOS]`, `[EOS]`, and `[SEP]`, to indicate the start and end of sequence, and the separation between faces, respectively. During training, we order faces according to the edge indices and then concatenate all faces into a single sequence. Take Figure 2 as an example, the desired output sequence would be $(\texttt{SOS}, e_1, e_3, e_4, e_8, \texttt{SEP}, e_2, e_3, e_5, e_6, e_7, e_9, e_{11}, \texttt{SEP}, e_8, e_9, e_{10}, e_{12}, \texttt{EOS})$.

The second variant, *seq2seq + co-edge*, also generates all faces sequentially but uses co-edges. In this variant, the input sequence would include all the co-edges and three special tokens `[SOS]`, `[EOS]`, and `[SEP]`. We order the faces in the same way as the first variant. Thus, the desired output sequence for the example in Figure 2 is $(\texttt{SOS}, c_1, c_2, c_3, c_4, \texttt{SEP}, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, \texttt{SEP}, c_{12}, c_{13}, c_{14}, c_{15}, \texttt{EOS})$.

Table 2 shows quantitative results on the full test set. The seq2seq model using edges performs substantially worse because (i) unlike co-edges, edges do not encode directional information, so the model cannot take advantage of the natural order of face loops; and (ii) an edge is shared by two faces, so there is ambiguity about which face to predict given an edge. Comparing the two models using co-edges, the one with parallel prediction has slightly lower precision (-1.1%) but higher recall ($+5.4\%$). This is because it makes a prediction starting from each co-edge, thus would cover each face several times. In contrast, the model with sequential prediction only generates each face once. We also record the average runtime of the models for one shape in the test set. As one can see in Table 2, the model with parallel prediction is much faster.

Note that our focus here is on the various ways to detect face loops, thus we do not perform face type classification in this experiment. To enable joint face type classification for the two variants, one can simply replace the `[SEP]` token with the corresponding face type tokens. Based on our experience, enabling face type classification has negligible impact on the face identification results.

input data	precision	recall	cls. rate
Ours	93.8	95.9	97.5
Ours - perspective	93.6	96.2	97.6
Ours - fixed viewpoint	95.9	97.3	97.6

Table 3. Experiment results on input data.

Experiment on input data. In the next experiment, we study the performance of our model with different types of input data. First, we replace orthographic projection with perspective projection when generating the 2D line drawings. As shown in Table 3, this change has very small impact on all the metrics. Second, we fix the camera viewpoint to generate an isometric drawing for each shape. In CAD, an isometric view is commonly used to reveal as much information about the 3D shape as possible, and to avoid situations where the object's edges or vertices coincide (or appear as joined) accidentally. Therefore, such a viewpoint is considered easier than random viewpoints. By employing a fixed viewpoint for all objects, our model achieves even higher accuracies.

Furthermore, our model achieves high (and almost identical) classification rates with all input data types.

Qualitative results. Figure 4 visualizes *all* incorrect predictions made by our model for various objects. Some com-

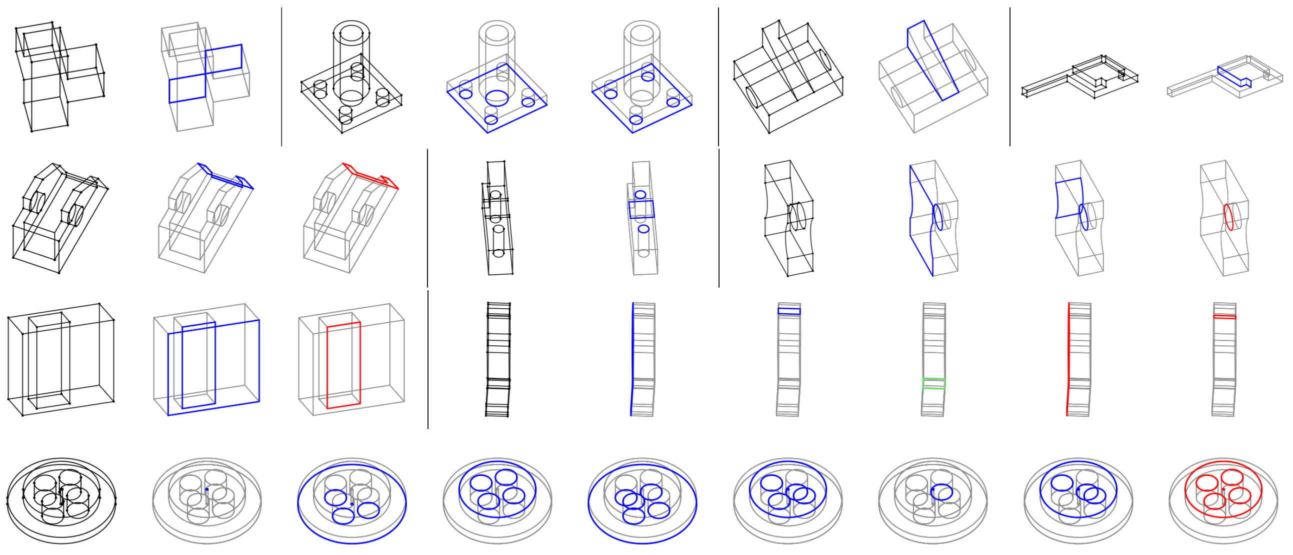

Figure 4. Incorrect predictions made by our face identification model. For each case, we show the input line drawing, followed by predictions with wrong face loops (blue), predictions with wrong face types (green), and missed faces (red).

mon problems are (i) incomplete prediction when a face consists of multiple nested edge loops, and (ii) grouping edges or loops which belong to different faces. But as we will see in the next section, these errors have varying impact on the reconstruction of the 3D models.

5. 3D Object Reconstruction

We now tackle the problem of 3D reconstruction with the predicted face loops and types. In the literature, this is often formulated as recovering the missing depths of the vertices of a line drawing. While knowledge about face topology significantly reduces the number of degrees of freedom, such information itself is not enough to uniquely determine the 3D geometry. Prior work [17, 25, 30] resort to additional optimization criteria such as MSDA, face planarity, line parallelism, and corner orthogonality. We observe a few problems with these approaches: (i) the criteria are designed to emulate the human perception of a 2D line drawing as a 3D object, but it is not uncommon for them to be violated in practice; (ii) the search for optimal solutions could get stuck at local minimum; (iii) they do not use information about face types.

Our primary goal is to build 3D models from the output of a deep face identification model – something never considered in prior work. Thus, we prefer a solution which decouples the impact of mistakes made by the network from that of other constraints or algorithms employed.

To this end, we develop a simple method that relies on only one type of constraints: line parallelism. As shown in Figure 5(a), we assume that there are three mutually orthogonal directions $\{\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3\}$ in the 3D scene, *i.e.*, $\mathbf{l}_j^T \mathbf{l}_k = 0, \forall j \neq k$. While these directions are provided *a priori* in this work, techniques for automatic estimation exist: for orthographic projections, these directions can be found by grouping parallel lines in a line drawing; for perspective projections, these directions correspond to the three dominant vanishing points. Afterward, the 3D directions $\{\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3\}$ can be obtained via camera calibration [11].

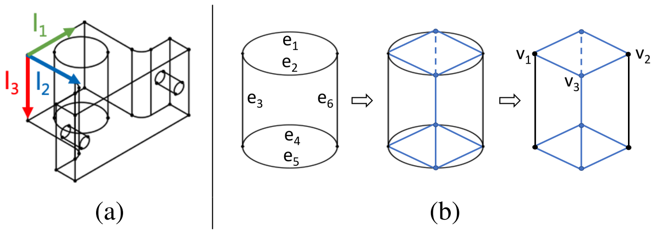

Figure 5. 3D reconstruction with predicted faces. (a) Illustration of three dominant directions. (b) Handling curved surfaces.

With this assumption, we are able to align the faces to the dominant directions according to their enclosing edges and solve for the 3D geometry via convex optimization. Below we first describe our method for objects with planar faces, then extend it to curved surfaces.

5.1. 3D Reconstruction of Planar Objects

Suppose we are given an object with M faces $F = \{f_1, \dots, f_M\}$ and L vertices $V = \{v_1, \dots, v_L\}$. For each vertex, we write $v_l = [x_l, y_l, z_l]^T$, where z_l is the unknown depth of v_l in the camera coordinate system.

If the object is planar, then each face f_i can be represented by the plane equation $a_i x + b_i y + z + c_i = 0$. If a vertex v_l lies on two or more faces, for each pair of faces, say (f_1, f_2), we have $a_1x_l + b_1y_l + c_1 - a_2x_l - b_2y_l - c_2 = 0$. For all vertices in V, we can write similar constraints for pairs of faces. Rewriting all these linear equations in matrix form, we have: $\mathbf{P}_1\mathbf{f} = 0$, where $\mathbf{f} = [a_1, b_1, c_1, \ldots, a_M, b_M, c_M]^T$ is a vector consisting of all the parameters of the faces.

For each dominant direction $\mathbf{l}_j, j = \{1, 2, 3\}$, we can identify all edges in the line drawing which are parallel to it, thus also find the faces which align with it. For such a face, say f_1, we have $a_1l^j_x + b_1l^j_y + l^j_z = 0$. Rewriting all these linear equations in matrix form, we have: $\mathbf{P}_2\mathbf{f} = 0$.

Combining the above constraints, we can find $\mathbf{f}$ by solving a convex optimization problem:

$$\begin{aligned} \min_{\mathbf{f}} \quad & \|\mathbf{Pf}\|_1, \\ \text{s.t.} \quad & z_l = -(a_ix_l + b_iy_l + c_i) > 0, \forall \mathbf{v}_l \text{ on } f_i. \end{aligned} \tag{4}$$

Here, the constraints enforce that the 3D vertices lie in front of the camera.

5.2. Handling Curved Surfaces

To deal with curved surfaces, we follow the approach proposed in [30]. The main idea is to replace a curve with straight line segments so that the object becomes a polyhedron. Then, methods for planar objects, such as the one described in Section 5.1, can be applied. Finally, the curved surface is recovered by fitting general Bézier curves to the corresponding 3D vertices.

In our case, because the face types are known, the original approach can be substantially simplified, as we (i) do not need to employ an algorithm to distinguish between curved and planar faces; (ii) can develop face approximation and fitting methods for each specific face type.

Figure 5(b) shows an example of converting a curved surface on a cylinder into planar faces. For face (e_2, e_3, e_5, e_6), we find the singular points on curves e_2 and e_5 and then replace each curve with two straight lines. Once the 3D geometry of the polyhedron is reconstructed, the original curves can be recovered by fitting a 3D circle to the three vertices (*e.g.*, v_1, v_2, and v_3 in the figure).

5.3. Experiments

Before presenting the 3D reconstruction results based on the predicted faces, we point out that in rare cases ($< 5\%$) the reconstruction may not be perfect even with ground truth faces, due to the simple assumption (*i.e.*, line parallelism) we employ in the pipeline. Figure 6 shows two examples where major parts of the model do not align with any dominant direction, making the reconstruction under-constrained. As a result, the 3D model may appear to be distorted (first row) or partly incorrect (second row). In practice, these may be addressed by introducing additional constraints. But we choose to keep the reconstruction method simple so that the impact of errors in face identification can be better analyzed.

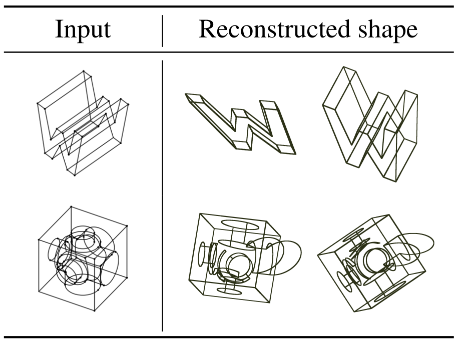

Figure 6. Two problematic cases of 3D reconstruction with ground truth faces.

Figure 7 shows 3D reconstruction results for various shapes using the predicted faces. In the *first row*, we show two cases in which all faces are correctly identified and the 3D model is fully reconstructed. As a comparison, we also train AtlasNet [9], a popular deep learning method for single-view 3D reconstruction, on our dataset and include the test results in Figure 7. Unlike our method, most existing deep learning methods take an image as input and generate unstructured point cloud or meshes in an end-to-end fashion. As one can see, such a method struggles to learn with the wireframe inputs because the features are very sparse when treated as an image. In contrast, we propose to detect structures (*i.e.*, face topology) in the wireframe and use geometric reasoning for 3D inference. Therefore, our method is able to output clean, structured 3D models.

The *second row* and *third row* of Figure 7 show examples in which our face identification model makes some incorrect predictions (see Figure 4) but the 3D reconstruction results are unaffected. For the two cases in the second row, our model generates edge loops which are not part of the true face topology, but all included edges fall onto the same surface (*e.g.*, a plane). For the two cases in the third row, the incorrect faces are filtered because they align to more than two dominant directions. Since the number of constraints created by the faces is typically larger than the number of unknowns in Eq. (4), the 3D model can be recovered even if some faces are missed.

The *fourth row* shows two examples in which incorrectly predicted faces affect local part of the 3D model. And the *fifth row* shows two examples in which 3D reconstruction completely fails and the recovered shapes appear flat. The latter typically occurs when our model makes multiple mistakes (*e.g.*, grouping edges on the opposite sides of the 3D shape into one face). Again, we refer readers to Figure 4 for visualization of the mistakes made by our model.

In the *last row* of Figure 7, we show two interesting cases in which even humans have trouble inferring the 3D geom-

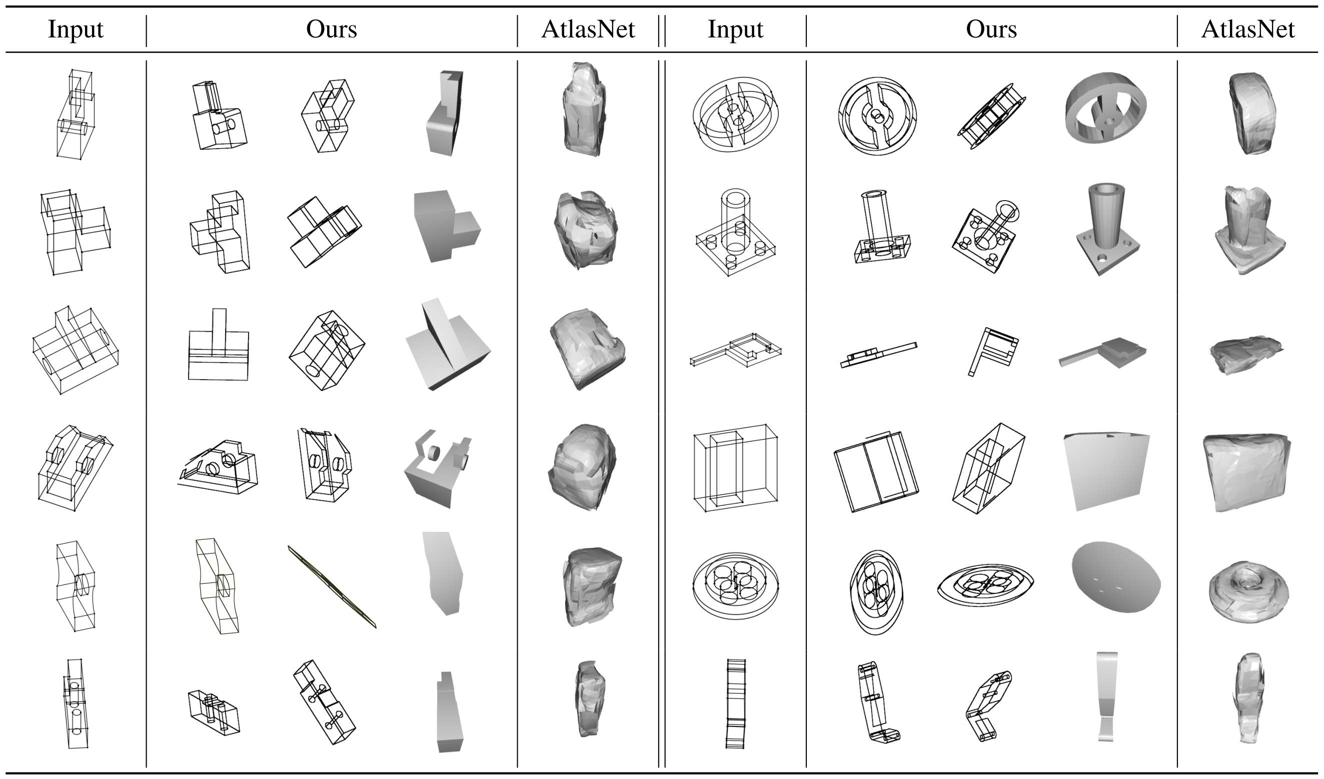

Figure 7. 3D reconstruction results. For our method, we show two different views of the reconstructed 3D wireframe, plus the mesh from the same viewpoint as the input. For AtlasNet, we show the mesh from the same viewpoint as the input.

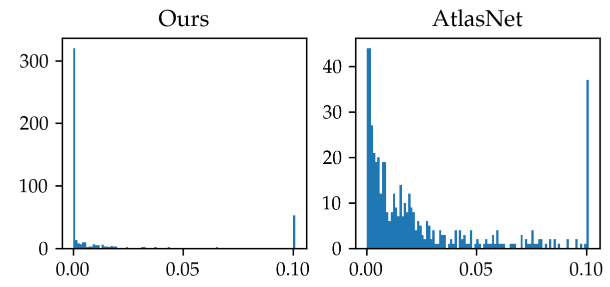

Figure 8. Histogram of Chamfer distances between our predicted meshes and the ground-truth meshes on 504 objects in test set.

etry due to the chosen viewpoints. In contrast, our method does a decent job using the extracted face loops. This suggests that our method is not sensitive to the viewpoints.

Quantitative results. We compute the Chamfer distance between the predicted meshes and the ground truth in the test set, and plot the distributions in Figure 8. As one can see, the structured 3D models obtained by our method tend to be more accurate. For example, for more than 60% of the objects, our method achieves a distance of $< 10^{-3}$.

6. Discussion

Limitations. In this work, we present a data-driven approach to face identification. We point out that the proposed method should not be treated as a replacement or competitor to the geometry- and topology-based methods. Instead, we have found that our method complements existing techniques in several aspects, such as capturing the intent of designers, handling curved surfaces and disjoint components in the edge-vertex graph. Meanwhile, our model could make wrong predictions, while geometry- and topology-based methods are guaranteed to succeed when all assumptions are met.

Our method assumes that input 2D wireframe is clean and noise-free. In a practical system (*e.g.*, SMARTPAPER [23]), this may be the outcome of multiple sketch cleaning and beautification steps. We hypothesize that our deep model can be made robust to noisy inputs with proper training (*e.g.*, data augmentation), but a thorough investigation is beyond the scope of this paper.

Future directions. Our work opens up several directions for future work. One direction we are particularly interested in is identifying conflicting constraints (due to incorrect predictions made by a deep network) in a geometric constraint system for 3D reconstruction. In this work, we treat all predicted faces equally. A better solution will not only improve the 3D reconstruction results, but may also generalize to other types of structural constraints in CAD.

References

[1] Fatemeh Abbasinejad, Pushkar Joshi, and Nina Amenta. Surface patches from unorganized space curves. *Comput. Graph. Forum*, 30(5):1379–1387, 2011. 2

[2] Siddarameshwar Bagali and Warren N. Waggenspack. A shortest path approach to wireframe to solid model conversion. In *ACM Symposium on Solid Modeling and Applications*, pages 339–350. ACM, 1995. 2

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Int. Conf. Learn. Represent.*, 2015. 2

[4] John A. Brewer and S. Mark Courter. Automated conversion of curvilinear wire-frame models to surface boundary models; a topological approach. *SIGGRAPH Comput. Graph.*, 20(4):171–178, 1986. 2

[5] Ayan Das, Yongxin Yang, Timothy Hospedales, Tao Xiang, and Yi-Zhe Song. Béziersketch: A generative model for scalable vector sketches. In *Eur. Conf. Comput. Vis.*, pages 632–647, 2020. 4

[6] Ronald D. Dutton and Robert C. Brigham. Efficiently identifying the faces of a solid. *Comput. Graph.*, 7(2):143–147, 1983. 2

[7] Fen Fang, Yong Tsui Lee, and Mei Chee Leong. Identification of faces in line drawings by edge decomposition. *Pattern Recognit.*, 48(12):3825–3842, 2015. 1, 2

[8] Mark A. Ganter and John J. Uicker. From wire-frame to solid geometric: automated conversion of data representations. *Computer in Mechanical Eng.*, 22(2):40–45, 1983. 2

[9] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 216–224, 2018. 7

[10] Patrick M. Hanrahan. Creating volume models from edge-vertex graphs. *SIGGRAPH Comput. Graph.*, 16(3):77–84, 1982. 2

[11] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000. 6

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997. 3

[13] Pradeep Kumar Jayaraman, Aditya Sanghi, Joseph G. Lambourne, Karl D. D. Willis, Thomas Davies, Hooman Shayani, and Nigel Morris. Uv-net: Learning from boundary representations. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 11703–11712, 2021. 2

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2015. 4

[15] 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. 2, 4

[16] Joseph G. Lambourne, Karl D. D. Willis, Pradeep Kumar Jayaraman, Aditya Sanghi, Peter Meltzer, and Hooman Shayani. Brepnet: A topological message passing system for solid models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12773–12782, 2021. 2

[17] Jianzhuang Liu, Liangliang Cao, Zhenguo Li, and Xiaoou Tang. Plane-based optimization for 3D object reconstruction from single line drawings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(2):315–327, 2008. 2, 6

[18] Jianzhuang Liu, Yong Tsui Lee, and Wai-Kuen Cham. Identifying faces in a 2d line drawing representing a manifold object. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(12):1579–1593, 2002. 1, 2, 3

[19] Jianzhuang Liu and Xiaoou Tang. Evolutionary search for faces from line drawings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(6):861–872, 2005. 1, 2

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

[21] 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. 2

[22] Wamiq Reyaz Para, Shariq Farooq Bhat, Paul Guerrero, Tom Kelly, Niloy J. Mitra, Leonidas J. Guibas, and Peter Wonka. Sketchgen: Generating constrained CAD sketches. In *Adv. Neural Inform. Process. Syst.*, 2021. 2

[23] Amit Shesh and Baoquan Chen. SMARTPAPER: an interactive and user friendly sketching system. *Comput. Graph. Forum*, 23(3):301–310, 2004. 8

[24] Moshe Shpitalni and Hod Lipson. Identification of faces in a 2d line drawing projection of a wireframe object. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(10):1000–1012, 1996. 1, 2

[25] Moshe Shpitalni and Hod Lipson. Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Comput. Aided Des.*, 28(8):651–663, 1996. 2, 6

[26] Moshe Shpitalni and Hod Lipson. Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *J. Mech. Des*, 119(1):131–135, 1997. 3

[27] Peter A.C. Varley and Pedro P. Company. A new algorithm for finding faces in wireframes. *Comput. Aided Des.*, 42(4):279–309, 2010. 1, 2, 3, 4, 5

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inform. Process. Syst.*, pages 5998–6008, 2017. 2, 3, 4

[29] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Adv. Neural Inform. Process. Syst.*, pages 2692–2700, 2015. 2, 3

[30] Yingze Wang, Yu Chen, Jianzhuang Liu, and Xiaoou Tang. 3D reconstruction of curved objects from single 2d line drawings. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1834–1841, 2009. 2, 6, 7

[31] 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. 2

[32] 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. 2

[33] 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. 2

[34] Yixin Zhuang, Ming Zou, Nathan Carr, and Tao Ju. A general and efficient method for finding cycles in 3d curve networks. *ACM Trans. Graph.*, 32(6):180:1–180:10, 2013. 2