Dynamic Furniture Modeling Through Assembly Instructions

Tianjia Shao[*] Dongping Li[*] Yuliang Rong[*] Changxi Zheng[†] Kun Zhou[*][‡]

[*]State Key Lab of CAD&CG, Zhejiang University [†]Columbia University

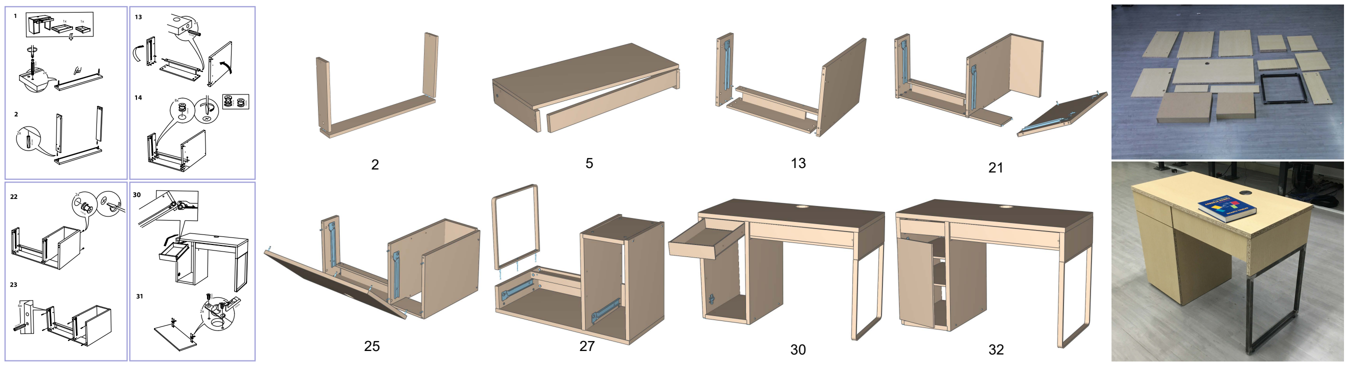

Figure 1: *Starting from a multi-step furniture assembly instruction (with selected steps shown on the left), we reconstruct both the 3D shapes of furniture components and their dynamic assembly process (with selected snapshots in the middle). The recovered 3D shapes can further be edited for physical fabrication (right). All the diagrams here were redrawn by our artists from an* Ikea *assembly instruction.*

Abstract

We present a technique for parsing widely used furniture assembly instructions, and reconstructing the 3D models of furniture components and their dynamic assembly process. Our technique takes as input a multi-step assembly instruction in a vector graphic format and starts to group the vector graphic primitives into semantic elements representing individual furniture parts, mechanical connectors (e.g., screws, bolts and hinges), arrows, visual highlights, and numbers. To reconstruct the dynamic assembly process depicted over multiple steps, our system identifies previously built 3D furniture components when parsing a new step, and uses them to address the challenge of occlusions while generating new 3D components incrementally. With a wide range of examples covering a variety of furniture types, we demonstrate the use of our system to animate the 3D furniture assembly process and, beyond that, the semantic-aware furniture editing as well as the fabrication of personalized furnitures.

Keywords: Assembly instructions, furniture modeling, supervised learning, personalized fabrication

Concepts: •**Computing methodologies** → **Shape modeling;** *Parametric curve and surface models;*

[*]tianjiashao@gmail.com, kunzhou@acm.org
[†]cxz@cs.columbia.edu
[‡]Corresponding author

SA '16 Technical Papers, December 05-08, 2016, Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2982416

1 Introduction

Furniture comes with assembly instructions, a series of schematic diagrams intended to visually assist the customer's furniture assembly process. Often composed of multiple pages, assembly instructions depict a step-by-step, dynamic process of assembling individual furniture pieces into a complete and functional body (see Figure 2). Today, they have become an integral part of furniture products, with a wide access from the website of almost every brand in the furniture industry, such as IKEA, West Elm, Nitori, Room&Board, to name a few.

Along with the increasing functionality and also complexity of modern furnitures is the increasingly lengthy instructions. Oftentimes, they present numerous furniture pieces, subtly different screws and other mechanics, and detailed insets, with changing view angles and projection scales across steps. Indeed, when assembling a furniture, a user has to constantly refer instruction steps back and forth and can easily lose track. Arguably, an assembly instruction could only embody discrete assembly "snapshots", but could not teach it, nor even describe a continuous and fluent assembly process fully.

We propose a method to convert a multi-step furniture assembly instruction into a 3D, continuous flow of assembly process. With a few user interactions, our technique parses the assembly instruction, constructs 3D models of furniture parts and mechanic connectors, and more notably understands the dynamic process of furniture assembly.

Understanding the 3D furniture models and their dynamic assembly opens the door to new applications. We demonstrate three of them: First, we present the user a 3D animated assembly processing, which allows for arbitrary playback from a user-controlled view angle. An animated instruction can complement the static assembly diagrams and enrich the visual expressiveness of an assembly instruction. Second, we infer the semantics of furniture components from the dynamic assembly process, and allow the user to edit furniture models in a semantic-aware way. Lastly, we show that the reconstructed 3D furniture models can be directly fabricated through woodwork or 3D digital fabrication, allowing for user customizations.

We formulate our problem not simply as one of 3D reconstruction, but one of *dynamic* 3D reconstruction. Provided a multi-step assembly instructions in vector graphic format (e.g., PDF files), we segment the vector graphics semantically and reconstruct 3D shapes of individual furniture components and mechanical connectors (e.g., screws and bolts). Further, we align these 3D shapes with compatible scales and orientations and infer the way of assembling them together. Throughout, significant challenges arise. Our algorithm needs to robustly handle rapid changes of view angles across steps, occlusions among furniture components, and incremental introduction of new furniture parts, and to establish cross-step correspondence of furniture parts as well as address the illustrative imprecisions of the assembly diagrams.

Techniques and contributions. We propose a supervised learning algorithm to group vector graphic primitives of an assembly diagram into semantic components. To reconstruct 3D shapes of individual furniture components, we formulate an optimization problem that estimates the vanishing points of vector graphic paths and leverages face extrusion. We then develop an algorithm to correctly align reconstructed 3D shapes for assembly. To this end, our algorithm analyzes the desired spatial relationships among furniture components and mechanical connectors and formulate another optimization problem to transform the 3D shapes. Next, we recover dynamic assembly actions by analyzing the placement of mechanical connectors and arrows in the assembly diagram. Last, we extend our algorithm to handle cross-step correspondence, to recognize previously assembled furniture parts from a new view angle.

Unlike pixel-image-based 3D reconstruction [Chen et al. 2013], our algorithm is tailored for processing vector graphic images, the standard format of almost all furniture assembly instructions. We evaluate our technique using real-life assembly instructions obtained from a number of furniture sites, including IKEA and Nitori. Our tests involve common furniture types, such as beds, chairs, benches and cabinets. Among all the tests, our algorithm succeeded with a high recognition accuracy (see §9), with a few failure cases that can be easily revised with simple user strokes.

2 Related Work

Parsing diagrams and sketches. There have been many work focusing on automatically parsing different types of diagrams and sketches such as engineering drawings [Haralick and Queeney 1982; Tombre 1998] and cartographic road maps [Mena 2003]. More recently, Berthouzoz et al. [2013] introduced an approach to parse sewing patterns to reconstruct the 3D model of a garment. Our goal is similar in spirit, but we focus on a different type of diagrams, the furniture assembly instructions. Unlike sewing patterns describing 2D developable surfaces, furniture assembly instructions depict 3D shapes. This difference leads to fundamentally different techniques. We need to address diagram segmentation, object recognition, and 3D shape reconstruction.

Numerous work have semantically segmented 2D hand-drawn sketches and diagrams. We refer to the comprehensive survey [LaViola et al. 2006] for an overview of this rich field. Early work that identifies low-level shapes like lines and arcs [Gennari et al. 2005] relies on *ad hoc* rules and domain-specific knowledge to group them semantically. Recent work also addresses freehand sketches via database retrieval: Huang et al. [2014] segmented and recognized free-hand sketches by searching for similar 3D components in a database; Sun et al. [2012] used a large set of clip art images as a knowledge base for segmentation. These methods are often limited by the scope of the database. We also use a database, but only for recognizing screws, bolts, and nuts — the mechanical connectors that are standardized, mass-produced, and reused. To recognize furniture main bodies, which have much more varieties than mechanical connectors, we do not rely on database. Instead, we formulate a graph labeling problem based on the connections of vector graphic elements.

High recognition accuracy has been achieved for such specific applications as mathematical equations [Jr. and Zeleznik 2004] and chemical drawings of molecular structures [Ouyang and Davis 2011]. Eitz et al. [2012a] studied "How do humans sketch objects?", which is the first large-scale analysis of human sketches. In parallel, many methods focus on real-time shape retrieval using shape features [Funkhouser et al. 2003; Eitz et al. 2012b; Xu et al. 2013] and shape matching [Shao et al. 2011]. Unlike free-drawn sketches, furniture assembly diagrams are regular. Thus, our system combines sketch recognition for furniture components and shape retrieval for mechanical connectors, resulting in a high recognition accuracy (§9.1).

Sketch based 3D modeling. In addition to sketch recognition, we also reconstruct 3D shapes of furnitures. In general, sketch-based 3D modeling methods aim to generate 3D lines and curves from 2D sketches and in turn reconstruct shapes. Among the early work, Zeleznik et al. [2006] used sketches to model 3D man-made shapes; Igarashi et al. [1999] modeled 3D shapes using stroke inflation; and Karpenko and Hughes [2006] exploited cusps and T-junctions in sketches for 3D modeling; Chen et al. [2008] generated 3D polyhedrons from a single-view sketching interface, exploiting simple heuristics and optimization to estimate 3D positions of 2D points of sketches. Another approach, ILoveSketch [Bae et al. 2008], allows to use sketches to create curved shapes by exploiting shape symmetry and two-view epipolar geometry. Later, Gingold et al. [2009] developed an interface for sketch-based 3D modeling by placing 3D primitives and annotations, while Schmidt et al. [2009] presented an analytic drawing method to lift 2D curved sketches to 3D curves using geometry constraints derived from 3D scaffolding. In recent work [Xu et al. 2014; De Paoli and Singh 2015], 3D curves and models have been correctly reconstructed without the need of 3D scaffolds.

The most related work to ours are the component-based modeling methods [Zheng et al. 2012; Shtof et al. 2013; Shao et al. 2013; Chen et al. 2013; Cao et al. 2014]. These methods start from interactively fitting 3D primitives (e.g., generalized cylinders) to 2D drawings and contours of individual parts of an object; then they optimize the 3D models subjecting to geo-semantic constraints. Our work focuses on furniture components, which are primarily vertical cuts from boards and thus can be approximated as extruded shapes [Cao et al. 2014]. Further, we incorporate mechanical connectors and infer the indicated semantic relations to correctly align furniture assembly.

Furniture design and fabrication. Computer graphics has witnessed increasing interests on furniture design and fabrication. In a pioneering work, Agrawala et al. [2003] described a method to automatically generate assembly instructions for a variety of general objects, including furnitures. Our method addresses an inverse problem, inferring 3D models and dynamic assembly process from 2D instructions. Related to fabrication, Lau et al. [2011] generated fabricatable furniture parts and connectors using a grammar-based method. Saul et al. [2011] presented an interactive system for sketching chair models that can be fabricated. To guide a structurally sounding furniture design Umetani et al. [2012] analyzed physical stability and torque limits during a design process. Later, Schulz et al. [2014] used an interactive system to design 3D models by examples. Koo et al. [2014] and Rong et al. [2016] focused on creating works-like prototypes from functional specifications. Recently, Li et al. [2015] developed an algorithm to design foldable furnitures to save space. All these previous methods analyze and optimize input

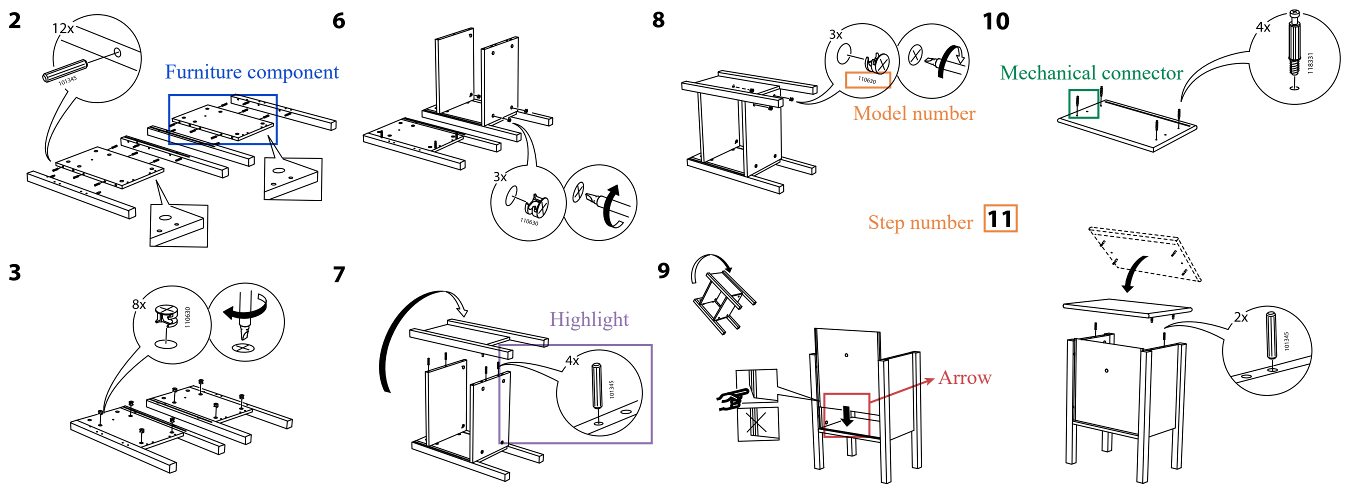

Figure 2: Selected instruction steps *of a bedside table. Typical furniture assembly instructions consist of a sequence of steps with furniture components incrementally depicted. Furniture components, mechanical connectors, arrows, numbers and highlights are the key visual semantic elements in the steps. Changes of furniture configuration (Step 6-7) and view angle (Step 8-9) and a switch of different furniture parts (Step 9-10) are all common in an instruction.*

3D shapes while aiming for 3D fabrication, our work focuses on generating 3D shapes from 2D diagrams, thus enabling furniture customization and fabrication from 2D assembly instructions (§10).

Analyzing instructions. Many psychology studies have analyzed various types of instructions, including the use of blobs, arrows [Tversky et al. 2000], animations [Tversky et al. 2002], and the design principles of assembly instructions [Heiser et al. 2004]. Generating interactive and dynamic content based on the analysis of instructions and drawings has been explored in other specific domains, such as exploded view diagrams [Li et al. 2004], augmented realities [Mohr et al. 2015; Gupta et al. 2012], interactive instruction tools [Zauner et al. 2003], illustration of mechanisms [Mitra et al. 2010], and 2D line drawings [Fu et al. 2011]. We focus specifically on the analysis of furniture assembly instructions. The reconstructed dynamic assembly process is useful not only as an animated instruction but also for other applications such as semantic furniture editing and personalized furniture fabrication.

3 Furniture Assembly Instructions

Numerous furniture assembly instructions are publicly available from a number of online sites, such as IKEA and Nitori [1], as downloadable vector graphic files in PDF format. While the instructions of different sites differ slightly in terms of visualization styles, they all use similar diagrammatic elements and follow the same visual flow. We examine these instructions and summarize here their graphic characteristics that we will exploit in our method.

Visual elements of assembly instructions. Typically, assembly instructions start in the first page with a list of standardized small components such as screws, bolts, hinges and other mechanical connectors (Figure 2). To help the user identify those connectors, associated with the graphic depiction are their model numbers. In subsequent steps, a series of diagrams depict how individual parts of the furniture are assembled incrementally toward a complete furniture body. In these diagrams, we identify five types of commonly used visual elements:

[1]Numerous furniture assembly instructions are downloadable from websites such as http://www.ikea.com and http://www.nitori-net.jp.

1. **Furniture components** depict shapes of a furniture's major parts that the user needs to work on at each step of an instruction. For instance, a simple table comprises a top, four legs, and some stretchers. In our method, we need to reconstruct 3D shapes of these components as well as the dynamic action to assemble them together.

2. **Mechanical connectors** (e.g., screws and hinges) are used to hold furniture components together. These connectors are drawn near their expected locations on the furniture. Oftentimes, there are also holes on the furniture to indicate their placement and dotted lines to specify their orientations and insertion directions. In our system, connectors provide important clues for correctly aligning furniture components.

3. **Arrows** indicate how one furniture component is attached to another component. They are the key to inferring the dynamic assembling process.

4. **Highlights** are insets of the steps, showing detailed shapes and model numbers of connectors. Together with the list of mechanical connectors in the first page, the highlights instruct the user exactly which mechanical connector should be used at a specific step and location.

5. **Numbers** in the instructions mainly appear as step numbers to indicate the order of assembly steps and as model numbers to indicate the type of mechanical connectors.

All these visual elements are depicted using vector graphic primitives, such as straight lines, polylines, polygons and Bézier curves. A visual element often consists of many different primitives, some of which may be redundant (e.g., overlapping with each other) or disconnected. As a result, a key challenge of our work is to infer the semantics of these low-level primitives and recognize high-level visual elements (§5).

Incremental instructions across steps. Visual elements are organized into a sequence of assembly steps. Over the steps, the furniture gets assembled progressively: every step of diagram introduces some new furniture components and mechanical connectors, which are either assembled as a new part of the furniture or attached to a previously built furniture part (e.g., see step 10 and 11 in Figure 2). More notably, to present key assembly steps clearly, the diagrams

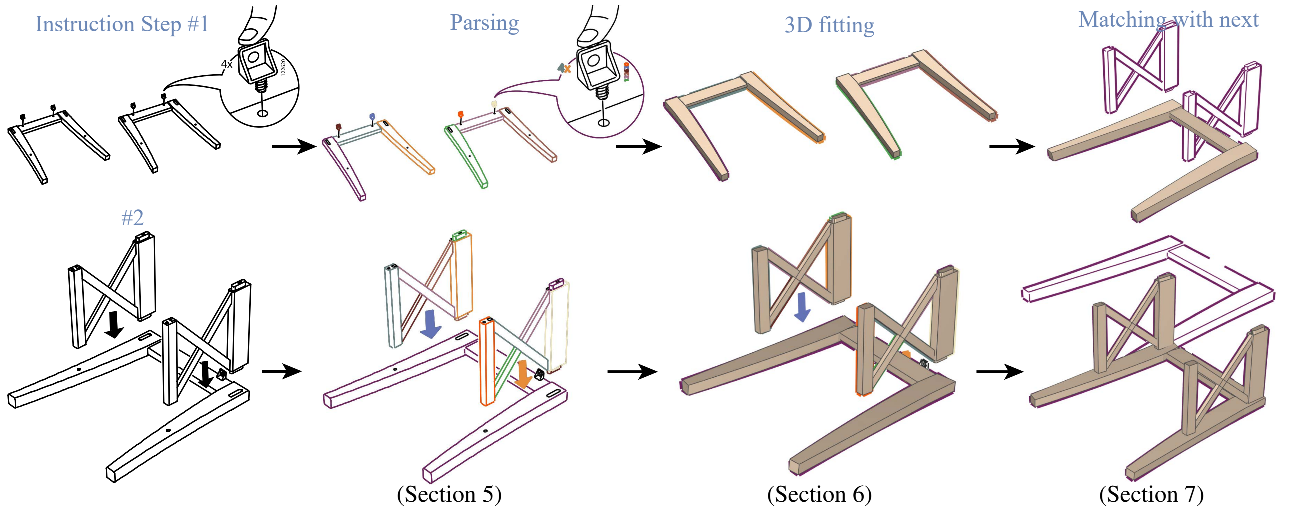

Figure 3: System overview. *Our system consists of three major stages: 2D parsing, 3D shape fitting, and establishing cross-step correspondence, as illustrated with two furniture examples here. Please refer to Section 4 for an outline of each stage.*

vary the perspective view angles across different steps (e.g., see step 8 and 9 in Figure 2). Thus, provided a new step of instruction, our method must recognize previously built furniture parts and newly introduced components from a possibly new view angle.

Imprecisions of assembly diagrams. We have observed that the view angles of each furniture component differ even in a single step, probably because of the imprecisions of drawing introduced by the illustrators. After all, most assembly instructions accessible online are not computer-generated (unlike [Agrawala et al. 2003]). This imprecisions exist widely in almost every instruction, introducing a significant challenge for reconstructing 3D furniture models reliably. To overcome this challenge, we seek to infer furniture constraints and cast the 3D reconstruction into a constrained global optimization problem (§6).

4 Overview

Our system takes as input an assembly instruction consisting of multiple steps of assembly diagrams. The pipeline proceeds in three major stages. It starts by grouping vector graphic primitives and classifying each group as one of the five types of visual elements introduced in §3. We formulate the grouping stage as a Markov Random Field problem, while addressing the recognition using a supervised learning algorithm (§5). Next, our method fits 3D models for individual furniture components and retrieves mechanical connectors from a database. We optimize their positions, orientations, and geometric sizes by taking into account assembly and geo-semantic constraints (§6). To handle incremental instructions across multiple steps, we identify the furniture parts that are depicted in a new step but built in a previous step. We reuse the previously reconstructed 3D models to help overcome the challenges of occlusions when reconstructing new 3D components (§7). The dynamic assembly process is inferred by analyzing the placement of mechanical connectors and arrows, and cross-step correspondence. Figure 3 illustrates these stages.

User interaction. Our machine learning algorithm, while having a high accuracy, may fail to group or recognize some visual elements occasionally. Throughout our pipeline, we allow the user to interactively correct path grouping, component recognition, and cross-step correspondence (§8). As shown in the video and reported in §9, the user interaction needed in practice is light and straightforward.

5 Parsing Instructions

We now present our algorithm of parsing visual semantic elements in assembly diagrams. With an instruction step loaded from a PDF file, the input to this algorithm is a set of vector graphic primitives, including lines, polygons and Bézier curves. We refer them all as *paths*. Our goal is to group the paths into one of the five types of visual elements summarized in §3.

At first thought, it is tempting to apply semantic segmentation techniques such as K-means clustering, nearest-neighbor assignment and semantic texton forests [Shotton et al. 2008], as all have proven successful for pixel images. However, unlike pixels which carry local information (i.e., color) for labeling, a vector graphic primitive can occupy a large region with a complex shape. It is insufficient to group paths purely based on local information. Instead, our parsing takes two stages, *segmentation* (§5.1) and *recognition* (§5.2): we first separate all the paths into groups, and then recognize which one of the visual element types each group represents. Both stages exploit machine learning algorithms.

Preprocessing. Before segmenting the paths, we preprocess them. We merge overlapping path segments and remove tiny segments (with a length less than 1pt). We also complete the paths that are weakly occluded by other vector graphic primitives (e.g. highlights occluded by numbers and furniture parts occluded by narrow bars): We first detect the paths whose endpoints are not connected with other paths. Then, in a local region (20pt$\times$20pt) around such an endpoint, we look for another disconnected endpoint of another

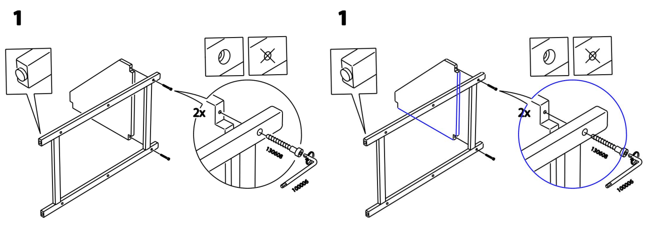

Figure 4: *Preprocessing of input vector graphic paths: we connect paths that are weakly occluded.*

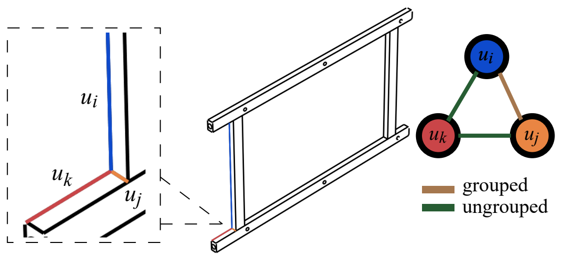

Figure 5: *A graph is built from vector graphics primitives. The paths (the blue, yellow and red curves) corresponding to u_i, u_j and u_k connect at the same point, but the graph edges (u_i, u_j), (u_i, u_k), (u_j, u_k) should be grouped differently, as they represent edges of different furniture parts.*

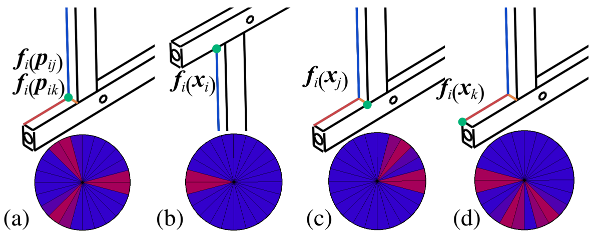

Figure 6: HO features *at different locations in a diagram. Features are computed in the local frame of u_i.* **(a)** *The graph edges (u_i, u_j) and (u_i, u_k) have the same local features at $\boldsymbol{p}_{ij}$ (here $\boldsymbol{p}_{ik}$ is the same point as $\boldsymbol{p}_{ij}$).* **(b-d)** *The context features at $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$, which are the far ends of paths connecting to $\boldsymbol{p}_{ij}$, help distinguish (u_i, u_j) from (u_i, u_k).*

path. If both paths are almost collinear locally near the endpoints[2], we join the two endpoints and merge the paths (Figure 4).

5.1 Segmenting Vector Graphic Primitives

Rationale. We propose a supervised learning algorithm to segment vector graphic paths into semantic groups, motivated by the following observations: (i) The visual elements differ semantically—for instance, furniture components and connectors depict geometric shapes while arrows indicate a dynamic process. These differences preclude the use of shape retrieval from a database [Huang et al. 2014] or procedural rules for grouping primitives [Gennari et al. 2005]. (ii) Unlike other types of vector graphic diagrams (e.g., sewing patterns [Berthouzoz et al. 2013]), furniture assembly instructions have no text labels to help segmentation. (iii) But the 2D drawing of assembly instructions are regular (in comparison to free-drawn sketches), suggesting that a small training set for the supervised learning can suffice.

Graph representation of primitives. We group the vector paths by formulating a graph labeling problem. First, we represent all the paths of an assembly diagram using a graph $G := (\mathcal{V}, \mathcal{E})$, where each node in $\mathcal{V}$ represents a path, two nodes are connected if the corresponding paths are connected in the diagram (Figure 5). If two paths are not connected at any of their endpoints (e.g., only intersecting at a point on the paths), we do not add an edge between their nodes. Our goal here is to label each edge in $\mathcal{E}$ as either "grouped" or "ungrouped", indicating how the paths should be clustered into visual elements.

If the lengths of two connected paths are very short (i.e., less than 5pt), the edge between their corresponding nodes is directly labeled as "grouped". This is because connected short paths are exclusively used to represent screws, so these paths ought to be grouped. Additionally, if at a point there are only two paths connected, the corresponding edge is directly labeled as "grouped".

Features. We train a binary classifier to label each edge in $\mathcal{E}$ by extracting feature vectors of graph edges. To decide if two paths should be grouped, we consider two types of features: *local features* capture the local path distribution at their connection point $\boldsymbol{p}_{ij}$, while *context features* capture how the paths are connected with other non-local paths. All these features need to be translation- and rotation-invariant.

Consider two paths u_i and u_j connected at an endpoint $\boldsymbol{p}_{ij}$. We note that often there exist other paths that also connect with u_i and u_j at $\boldsymbol{p}_{ij}$ (see inset of Figure 5). A straightforward attempt of extracting local features is to rasterize a small region centered at $\boldsymbol{p}_{ij}$ and use the pixel image descriptors (e.g., SIFT [Lowe 1999] and GALIF [Eitz et al. 2012b] features). However, these descriptors, designed for pixel images, can not fully exploit curve orientation indicated by vector graphic paths. Instead, we propose a feature called *Histogram of orientations* (HO) to capture the local distribution of paths incident to a point. To ensure the translational and rotational invariance, this feature is associated with each individual path u_i and is computed in its local frame of reference on the 2D plane. A complete feature vector to classify an edge e_{ij} consists of a group of HO features.

Suppose that there are N paths incident to the point $\boldsymbol{p}_{ij}$; each has a tangential direction $\mathbf{t}_k, k = 1..N$ at $\boldsymbol{p}_{ij}$ (see an example in Figure 6). Two of these tangential directions are those of paths u_i and u_j, respectively. Without loss of generality, we assume $\boldsymbol{t}_1$ is the tangential direction of u_i. We first estimate the probability distribution of a path incident to $\boldsymbol{p}_{ij}$ with an angle θ relative to the direction $\boldsymbol{t}_1$ using a Gaussian Mixture,

$$p_i(\theta) = \frac{1}{A}\sum_{k=2}^{N} \exp\left(-\frac{(\theta - \phi(\boldsymbol{t}_k, \boldsymbol{t}_1))^2}{2\sigma_\theta^2}\right), \qquad (1)$$

where $\phi(\boldsymbol{t}_k, \boldsymbol{t}_1)$ indicates the angle between two directions $\boldsymbol{t}_k$ and $\boldsymbol{t}_1$, measured clockwise in the local frame of u_i, σ_θ controls the impact range of each incident path ($\sigma_\theta = 15^\circ$ in our examples), and A is a constant to normalize the probability distribution $p_i(\theta)$. Next, we simply quantize this probability distribution to obtain an HO feature of u_i at $\boldsymbol{p}_{ij}$: we evenly split all 2D directions $[0^\circ, 360^\circ]$ into 24 bins and compute their weights by integrating $p_i(\theta)$ over the corresponding range of each bin, resulting in an HO feature, a vector of length 24. We denote it as $\boldsymbol{f}_i(\boldsymbol{p}_{ij})$. Figure 6 visualizes four HO features with color-coded circular histograms, where the red color indicates a large value and blue color indicates zero.

Next, we compute context features, ones that describe the path connection in non-local regions. Consider two paths u_i and u_j connecting at $\boldsymbol{p}_{ij}$ while respectively having two other endpoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ distant from $\boldsymbol{p}_{ij}$ (Figure 6). We define the context feature of the edge e_{ij} as $\{\boldsymbol{f}_i(\boldsymbol{x}_i), \boldsymbol{f}_i(\boldsymbol{x}_j), \boldsymbol{f}_j(\boldsymbol{x}_i), \boldsymbol{f}_j(\boldsymbol{x}_j)\}$, a vector of length 96, where the former two HO features describe path connections of u_i and u_j at their remote endpoints, measured in the local frame of u_i; and the latter two HO features describe that in the local frame of u_j.

The context feature is of importance for resolving local ambiguity at $\boldsymbol{p}_{ij}$. For instance, as shown in Figure 6, two connection points $\boldsymbol{p}_{ij}$ and $\boldsymbol{p}_{ik}$ can have very similar local features while having different semantics. But accouting for the path connections at their remote ends can help distinguish these cases.

Lastly, we assemble the feature vector for the edge e_{ij} by concate-

[2]In practice, we check if the angle between the tangential directions at both endpoints are less then 15°

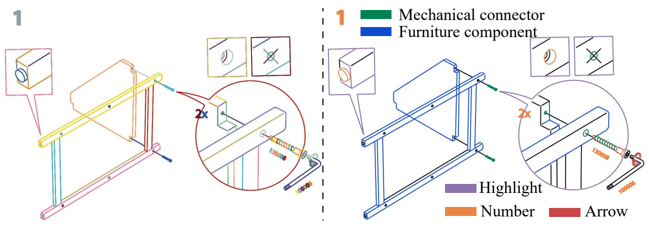

Figure 7: *(left) segmentation results with color-coded groups; (right) recognition results with color-coded visual semantic types.*

nating both local and context features:

$$\boldsymbol{f}_{ij} = \{\boldsymbol{f}_i(\boldsymbol{p}_{ij}), \boldsymbol{f}_j(\boldsymbol{p}_{ij}), \boldsymbol{f}_i(\boldsymbol{x}_i), \boldsymbol{f}_i(\boldsymbol{x}_j), \boldsymbol{f}_j(\boldsymbol{x}_i), \boldsymbol{f}_j(\boldsymbol{x}_j)\}. \quad (2)$$

Segmentation. We use the computed features to train a random regression forest [Fanelli et al. 2011]. Comparing to a standard random decision tree, the random regression forest is known to have the advantages of being efficient in both training and testing of large-scale data and being able to avoid over fitting.

During the test stage, provided an edge e_{ij} with a feature vector $\boldsymbol{f}_{ij}$, the random regression forest outputs the likelihood $P(e_{ij} = c)$ of the edge e_{ij} being labeled as c ($c = 0$ for "grouped" and $c = 1$ for "ungrouped"). To determine the binary labels from continuous likelihood values, we formulate a Markov Random Field problem, minimizing the following energy function over all possible binary labels:

$$\underset{\{e_{ij}=0/1\}}{\operatorname{argmin}} \sum_{\substack{e_{ij}\in\mathcal{E} \\ c=0,1}} E(e_{ij} = c) + \lambda \sum_{\substack{e_{ij},e_{jk}\in\mathcal{E} \\ c,d=0,1}} E(e_{ij} = c, e_{jk} = d). \quad (3)$$

Here $E(e_{ij} = c) = -\log P(e_{ij} = c)$ is the unary term measuring the cost of an edge being labeled as c, and $E(e_{ij} = c, e_{jk} = d)$ is the binary term penalizing when edges e_{ij} and e_{jk} are connected at u_j but assigned with different labels. We define this term as

$$E(e_{ij} = c, e_{jk} = d) = \begin{cases} \exp\left(-\frac{(l_i - l_q)^2}{2\sigma_l^2} - \frac{(\theta_i - \theta_q)^2}{2\sigma_\theta^2}\right), & \text{if } c \neq d. \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

where l_i is the path length of i and θ_i is unit orientation. σ_l is 10pt and σ_θ is 15° in our implementation. The binary term favors the same label for two edges e_{ij} and e_{jk} if the paths u_i and u_k have similar lengths and incident directions at the connection point. We solve this Markov Random Field problem using the standard graph cut algorithm [Boykov et al. 2001].

Grouping. After labeling all the edges, we remove the edges that are labeled as "ungrouped", and detect the connected components of the graph, naturally forming the paths into individual groups. Figure 7 illustrates one example of grouped paths.

5.2 Recognizing Visual Elements

We now recognize each group of primitives as one of the five types of visual elements, including furniture components, mechanical connectors, arrows, highlights, and numbers. For this classification problem, we again use the random regression forest method [Fanelli et al. 2011], but with features aiming to capture vector graphic patterns of an entire group.

Features. Inspired by the recent work on sketch-based shape retrieval [Eitz et al. 2012b], we use their GALIF features in our learning algorithm. Referring to that paper for more details, we outline the algorithm here and highlight our modifications in order to apply it to our problem. Given a group of vector graphic primitives, we rasterize it into a 128px×128px image. The GALIF feature describes a local patch (32px×32px) of this image. The selection of patches depends on the stage of the learning algorithm, whether it is for training or recognition (more details later). Here we present the algorithm operating on a single patch.

We start by applying a set of Gabor filters (6 in our case) to the patch, resulting in 6 filtered patches. The filter orientations are uniformly sampled over all 2D directions over $[0^\circ, 360^\circ]$ to capture the local sketch directions. Next, we split each filtered patch into 4×4 cells; each cell has a resolution of 8px×8px. We then average the color values in each cell, producing 6×4×4 scalars for a single patch. A GALIF feature is a vector stacking all these scalars. In addition, since a vector graphic image is scalable, we consider the relative size of the group by computing area of the bounding box of the primitive group relative to the page size. This number is put together with the GALIF feature to form the feature vector (with a length of 97) of a patch.

Training. We use the random regression forest for recognizing visual elements, again leveraging its ability of avoiding over fitting. During training, we rasterizing every group into a 128px×128px image, and then randomly sample 50 patches to compute their features for constructing the training set. Since the assembly instructions consist of only sparse paths, with no color filling inbetween, we avoid using patches in empty regions and only sample patches along vector graphic paths.

Recognition. To classify a given group of paths, we uniformly sample 100 patches from its 128px×128px rasterization. We use the decision trees in the trained random regression forest to classify every patch and aggregate the results. To classify a patch R using a single tree j, output from this decision tree is the likelihood function $P_j(c|R)$ of the patch R being labeled as a visual type c, where c is one of the five visual element types. The aggregated likelihood function $P(c|O)$ of a primitive group O being classified as type c is an average of the likelihood functions over all sampled patches from all decision trees,

$$P(c|O) = \frac{1}{K \times N} \sum_{i=1}^{N} \sum_{j=1}^{K} p_j(c|R_i),$$

where N is the number of sampled patches, R_i iterates through all patches, and K is the number of decision trees in the regression forest. Finally, the group is classified as a visual element type c with the largest likelihood.

Remark. After the recognition, we post-process two types of visual elements as special cases after they are recognized. (i) If a visual element is recognized as numbers, we further use Microsoft's OCR Library to identify the numerical values. Numbers in the diagram may be positioned along a tilted line. To improve OCR accuracy, we first level the numbers by finding the line across the centers of the numbers and aligning that line with x-axis. In our examples, this step helps to yield 100% OCR accuracy. (ii) Highlights are meant to show details (e.g. how to assemble the screws with screwdrivers, detailed shapes, connector numbers). Since the shapes can be reconstructed from the large scale diagram and we do not aim to illustrate how to assemble the screws, we safely discard all the highlights once they are recognized (but keep the numbers inside).

6 Dynamic 3D Reconstruction

After detecting visual elements in an assembly diagram, we now reconstruct 3D shapes. Among the five types of visual elements, only two represent 3D objects: the furniture components and mechanical connectors. We therefore focus on these two types in this section.

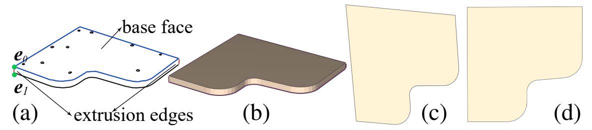

Figure 8: Extrusion. *Given a 2D diagram of a furniture part* **(a)***, we detect the extrusion edges and the base face (blue). Although using the vanishing point directly estimated using the extrusion edges, the fitted 3D shape matches well to the drawing* **(b)***, it fails to satisfy important geometric constraints such as the right angles at the corners of the board* **(c)***. In contrast, our optimized vanishing point helps to capture these shape regularities.*

Reconstruction of 3D mechanical connectors is straightforward, as almost all mechanical connectors are standardized and reused by many different furnitures. In assembly diagrams, the specific model of a mechanical connector is specified by text label, with both the connector and its label placed in a highlight (Figure 2). In light of this, we prepare 3D models of commonly used mechanical connectors and store them in a database. In practice, the database consists of 147 commonly used and manually modeled connectors, including different types of screws, bolts, studs, and hinges. During reconstruction, we detect the text in a highlight and use it to retrieve the 3D model of the labeled connector.

Much more challenging is the 3D construction of furniture components, as we need to not only reconstruct the 3D shapes of individual components but also infer their sizes and 3D positions and orientations so that they can be properly assembled. To this effect, we identify constraints indicated by mechanical connectors and arrows and optimize for the components' 3D shapes. Throughout, we also need to overcome imprecisions of the assembly diagrams—for example, the perspective projections of the components are often inaccurate, and even the view points are inconsistent.

Algorithm outline. Our algorithm has three stages: (i) we start from reconstructing 3D shapes of individual furniture components (§6.1); (ii) to assemble them together, we then optimize the positions and orientations of all components (§6.2); (iii) lastly, we reconstruct the dynamic assembly actions, extracting the directions and orientations of furniture components and mechanical connectors at each assembly step (§6.3). This algorithm operates on a single step of an assembly instruction. We will extend it to handle cross-step reconstruction in the next section.

6.1 Fitting Individual Parts

First, we reconstruct the 3D shapes of furniture components. Similar to the previous method [Cao et al. 2014], we hypothesize that the individual components are shapes extruded from 2D faces, called *base faces*. An *extrusion direction* is perpendicular to the base face, and its length is often short, resulting in thin board shapes (Figure 8). However, unlike their method, we aim to automatically detect base faces and extrusion directions, without user guidance. Briefly, we will first estimate camera settings used to project 3D shapes onto the 2D image plane, and then recover their base faces and extrusion directions in 3D.

Parallel edges and vanishing points. Our algorithm is built on two concepts, *parallel edges* in 3D and their *vanishing points* in 2D. An extruded 3D shape has two large, parallel faces, resulting in many pairs of parallel edges. After perspectively projected on a 2D plane, each pair of parallel edges, while remaining almost parallel, produces a vanishing point. Thus, we check all pairs of line segments of a furniture component. If two segments are nearly parallel (i.e., the cross product of their directions is below a threshold), we treat their 3D counterparts as parallel edges. Similar schemes of detecting parallel edges have also been used in image-based 3D reconstruction works (e.g., [Zheng et al. 2012]).

6.1.1 Camera Fitting

We use the well-established 3D vision theory to estimate camera parameters and refer to the textbook [Harltey and Zisserman 2006] for more details. A camera projection is described by its intrinsic matrix K and extrinsic matrix $[\mathsf{R}|\boldsymbol{t}]$. The former matrix captures the camera's focal length and image size, while the latter describes the camera's 3D orientation and location. Since we will estimate the relative positions of furniture components in the next stage (§6.2), here we can safely assume that the camera is axis aligned (i.e., $\mathsf{R} = \mathsf{I}_{3\times3}$) at the origin (i.e., $\boldsymbol{t} = \mathbf{0}$). Additionally, since the image size only determines the 3D shape's geometric size which is unimportant at this stage, we simply use a fixed camera image size. As a result, only the camera's focal length f needs to be estimated.

Using the 3D version theory (Result 8.22 on page 215 of [Harltey and Zisserman 2006]), one can estimate the focal length f from the positions of two vanishing points whose corresponding parallel lines in 3D are perpendicular to each other. In our problem, the extrusion direction is normal to the base face, and thus edges along the extrusion direction are perpendicular to edges of the base face (e.g., the red and blue lines in Figure 8). We detect all pairs of parallel segments in a 2D furniture component, and choose the shortest segment pairs as the component's *extrusion edges*, whose direction is the detected extrusion direction projected on 2D (Figure 8). The other parallel segments are edges on the base face, and thus perpendicular to the extrusion edges in 3D. Thereby, we obtain two vanishing points and use them to estimate the focal length f.

To handle the imprecisions of perspective projections, we use many pairs of vanishing points of a furniture component to compute f values and average them. Further, we assume that all furniture components in the same step are projected using the same camera settings (except insets inside highlights which are already ignored after parsing). So we estimate the camera parameters from individual furniture components and average them.

6.1.2 3D Shape Fitting

We now use the estimated intrinsic matrix K to reconstruct the base face and extrusion direction in 3D, and recover the 3D shape of a furniture component. Suppose a pair of parallel edges pointing along a 3D direction $\boldsymbol{n}$ has a 2D vanishing point $\boldsymbol{p}_v$. When the camera is axis-aligned at the origin, $\boldsymbol{n}$ and $\boldsymbol{p}_v$ are linearly related,

$$\boldsymbol{n} = \mathsf{K}^{-1}[\boldsymbol{p}_v^T \; 0]^T. \tag{5}$$

This relationship suggests a simple approach of estimating the 3D extrusion direction $\boldsymbol{n}_e$ using the vanishing point of detected extrusion edges. Unfortunately, because the extrusion edges are often very short, the resulting vanishing point is especially susceptible to projective imprecision, producing unnatural shape fitting (Figure 8). Instead, we estimate the vanishing point of extrusion edges by formulating an optimization problem.

Vanishing point optimization. Consider a vanishing point $\boldsymbol{p}_v$ and the computed 3D extrusion direction $\boldsymbol{n}_e$ using Eq. (5). Since the base face in 3D is perpendicular to $\boldsymbol{n}_e$, its 3D plane satisfying $\boldsymbol{n}_e \cdot \boldsymbol{x} = 0$ (up to a 3D translation, which is unimportant at this stage). Any line on the base face of 2D can be back-projected onto this 3D plane to compute their 3D directions. Our optimization strategy is to choose a vanishing point that preserves parallelism and orthogonality relations of the lines on the base face, after back-projecting them on the base face's 3D plane. To this end, we formulate an optimization problem,

$$\min_{\boldsymbol{p}_v} \left[w_p f_p(\boldsymbol{p}_v, \mathcal{S}_p) + w_o f_o(\boldsymbol{p}_v, \mathcal{S}_o) + w_e f_e(\boldsymbol{p}_v, \mathcal{S}_e) \right]. \qquad (6)$$

In this formulation, $f_p(\boldsymbol{p}_v, \mathcal{S}_p)$ enforces parallelism relations of every pair of detected parallel lines on the base face, defined as

$$f_p(\boldsymbol{p}_v, \mathcal{S}_p) = \frac{1}{|\mathcal{S}_p|} \sum_{\{\boldsymbol{s},\boldsymbol{t}\} \in \mathcal{S}_p} \|\boldsymbol{s} \times \boldsymbol{t}\|^2,$$

where $\mathcal{S}_p$ is the set of detected parallel line pairs on the base face; each pair, after back-projection, has the 3D directions $\boldsymbol{s}$ and $\boldsymbol{t}$. Similarly, the orthogonality term $f_o(\boldsymbol{p}_v, \mathcal{S}_o)$ is

$$f_o(\boldsymbol{p}_v, \mathcal{S}_o) = \frac{1}{|\mathcal{S}_o|} \sum_{\{\boldsymbol{s},\boldsymbol{t}\} \in \mathcal{S}_o} \|\boldsymbol{s} \cdot \boldsymbol{t}\|^2,$$

where $\mathcal{S}_o$ is the set of orthogonal 3D line segments detected on the 2D diagram. To construct $\mathcal{S}_o$, we first detect all Y-junctions connecting one segment (i.e., an extrusion edge) pointing along the detected extrusion direction and other two segments on the base face. We consider the latter two segments orthogonal to each other if the angle inbetween is in the range of $[60°, 120°]$. Lastly, the term $f_e(\boldsymbol{p}_v, \mathcal{S}_e)$ penalizes the deviation of the vanishing point from the detected extrusion direction, defined as

$$f_e(\boldsymbol{p}_v, \mathcal{S}_e) = \frac{1}{|\mathcal{S}_e|} \sum_{(\boldsymbol{e}_0,\boldsymbol{e}_1) \in \mathcal{S}_e} \left\| \frac{\boldsymbol{p}_v - \boldsymbol{e}_0}{\|\boldsymbol{p}_v - \boldsymbol{e}_0\|_2} - \frac{\boldsymbol{e}_1 - \boldsymbol{e}_0}{\|\boldsymbol{e}_1 - \boldsymbol{e}_0\|_2} \right\|_2^2, \qquad (7)$$

where $\mathcal{S}_e$ is the set of extrusion edges, and $\boldsymbol{e}_0$ and $\boldsymbol{e}_1$ are 2D endpoints of an extrusion edge (Figure 8-a).

We solve the optimization problem (6) using Matlab's fmincon function. The resultant vanishing point $\boldsymbol{p}_v$ determines the extrusion direction $\boldsymbol{n}_e$, which in turn defines the 3D plane of the base face (i.e., $\boldsymbol{n}_e^T \boldsymbol{x} = 0$). Next, we estimate the extrusion length h that translates the 3D plane to form another face of the 3D shape.

Extrusion length. We estimate h by formulating another optimization problem. As denoted in Eq. (7), suppose $\boldsymbol{e}_0$ and $\boldsymbol{e}_1$ are endpoints of an extrusion edge. Without loss of generality, we assume that $\boldsymbol{e}_0$ is on the base face, and $\boldsymbol{e}_1$ is on the extruded base face (Figure 8-a). We first back-project $\boldsymbol{e}_0$ on the 3D plane of $\boldsymbol{n}_e \cdot \boldsymbol{x} = 0$ and obtain a 3D point $\boldsymbol{v}_0$. Then its connected point $\boldsymbol{e}_1$ in 3D is $\boldsymbol{v}_0 + h\boldsymbol{n}_e$. This allows us to optimize h by solving

$$\min_h \frac{1}{|\mathcal{S}_e|} \sum_{(\boldsymbol{e}_0,\boldsymbol{e}_1) \in \mathcal{S}_e} \|\mathsf{P}(\boldsymbol{v}_0 + h\boldsymbol{n}_e) - \boldsymbol{e}_1\|_2^2,$$

where P is the camera projection matrix that projects a 3D point onto the 2D image plane, as estimated in §6.1.1.

Remark. It is possible that a furniture component has only one extrusion edge. This happens when that furniture component joins other furniture parts (e.g., see the middle bar connecting two legs in Figure 9-c). In this case, the problem (6) is ill-posed, as a single segment is insufficient to uniquely determine a vanishing point. Instead of solving (6), we first reconstruct the 3D shapes of connected furniture parts (e.g., the two legs in Figure 9-c). We then obtain the 3D extrusion direction $\boldsymbol{n}_e$ of the middle component by back-projecting its 2D extrusion edge onto its touching face of the 3D shape of the connected furniture part. Thereby, we sidestep the problem (6).

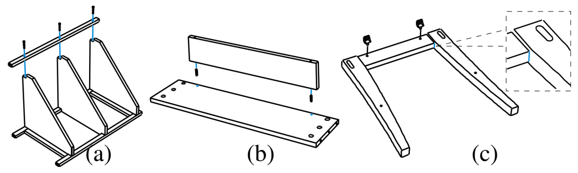

Figure 9: **(a)** *Connector constraint from a unidirectional screw;* **(b)** *connector constraint from a bidirectional screw;* **(c)** *component constraint, where two legs are connected with a bar.*

6.2 Handling Constraints

Now we assemble reconstructed 3D furniture components by properly translating, rotating and scaling them. This stage accounts for two levels of constraints. At a higher level, we ensure that the components are aligned correctly with compatible sizes and orientations by analyzing the placement of furniture components and mechanical connectors in 2D assembly diagram. At a lower level, we notice that furnitures often have regular shapes, which satisfy some geo-semantic relations—for example, some faces are coplanar, and legs have the same length. We further refine the 3D shapes by incorporating these constraints.

In this stage, the optimized variables are 3D rigid transformation together with nonuniform scales for all furniture components. We denote the 3D transformation of a component i using a matrix T_i, parameterized by 9 degrees of freedom (DoFs) including a translation $\boldsymbol{t}_i$ (3 DoFs), a rotation R_i (3 DoFs), and nonuniform scales $\boldsymbol{s}_i$ (3 DoFs). We jointly optimize all T_i using two sequential optimizations, starting with the one of ensuring assembly constraints, followed by the one with geo-semantic constraints.

6.2.1 Shape Transformation with Assembly Constraints

In an assembly diagram, there are two types of illustrations indicating assembly constraints.

Connector constraint. A mechanical connector connects two furniture components (Figure 9). In a diagram, its connection direction is indicated by a line, and its connection points are indicated by two points on both components respectively. We trace the line until we find 2D connection points located on two furniture faces. Since we have extracted the 3D shapes of both components i and j, we back-project the connection points to their located 3D faces (called touching faces) to obtain their 3D positions, denoted as $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$, respectively. With these positions, we define a cost function term as

$$E_a = \frac{1}{|\mathcal{H}|} \sum_{\{i,j\} \in \mathcal{H}} \left(\left\| \frac{\mathsf{T}_j\boldsymbol{h}_j - \mathsf{T}_i\boldsymbol{h}_i}{\|\mathsf{T}_j\boldsymbol{h}_j - \mathsf{T}_i\boldsymbol{h}_i\|_2} - \mathsf{T}_i\boldsymbol{n}_i \right\|_2^2 + \|\mathsf{T}_i\boldsymbol{n}_i - \mathsf{T}_j\boldsymbol{n}_j\|_2^2 \right),$$

where $\mathcal{H}$ denote the set of mechanic connectors that impose constraints; $\boldsymbol{n}_i$ and $\boldsymbol{n}_j$ are the 3D normal directions of the touching faces, determined after the 3D shape fitting stage (§6.1). Here the first term requires that the direction from $\boldsymbol{h}_i$ to $\boldsymbol{h}_j$ remains consistent with the touching face's normal direction after 3D transformation, while the second term requires the consistency between the two transformed normals (Figure 9-a). Sometimes, one of the two connection points can be occluded by a furniture component (Figure 9-b). To address this case, we first try to locate a connection point that is on the opposite but visible face and along the same connection direction. If such a point exists, we translate it to the occluded face to serve as the connection point. Otherwise, we simply use the middle point along the extrusion direction on the occluded face as the connection point.

Component constraint. Another type of constraints emerges when two furniture components are connected by another furniture component c (Figure 9-c). A typical example is the H-stretcher of a

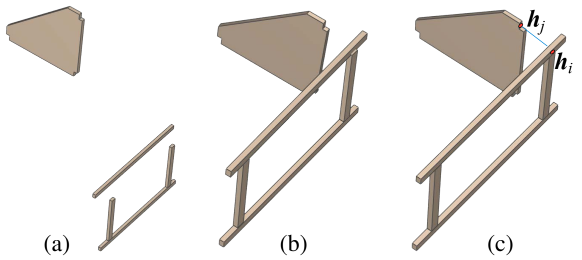

Figure 10: **(a)** *The initial 3D fitted shapes have incorrect positions, orientations and sizes;* **(b)** *the optimization using assembly constraints move the parts to correct positions;* **(c)** *these 3D shapes are further refined under geo-semantic constraints to improve shape alignment while retaining the match with the 2D drawing.*

bench in Figure 9. Oftentimes, in an assembly diagram, this type of connection is directly shown by putting three components closely together, without any depiction of mechanical connectors. We detect this configuration by checking if the component c has extrusion edges that are on the faces of the connected components, i and j. Putting all this kind of configurations in a set $\mathcal{C}$, we define a cost function term,

$$E_c = \frac{1}{|\mathcal{C}|} \sum_{\{i,j,c\} \in C} [\mathsf{T}_c \boldsymbol{e}_c \cdot (\mathsf{T}_i \boldsymbol{c}_i - \mathsf{T}_c \boldsymbol{c}_c)]^2 + [\mathsf{T}_c \boldsymbol{e}_c \cdot (\mathsf{T}_j \boldsymbol{c}_j - \mathsf{T}_c \boldsymbol{c}_c)]^2 ,$$

where $\boldsymbol{c}_i$, $\boldsymbol{c}_j$ and $\boldsymbol{c}_c$ are the (bounding-box) centers of component i, j, and c, respectively, and $\boldsymbol{e}_c$ is the extrusion direction of c. This constraint requires the plane formed by the transformed center points of i, j and c perpendicular to c's extrusion direction.

Regularization. In addition, we add a regularization term in the cost function. Consider a single furniture component i, whose vector graphic paths have a set of 2D endpoints, $\boldsymbol{p}^i_j, j = 1..N_i$. We compute their 3D positions $\boldsymbol{v}^i_j, j = 1..N_i$ via back-projecting $\boldsymbol{p}^i_j$ to their corresponding 3D faces computed in the shape fitting stage (§6.1). To ensure the 3D shapes holding together as indicated in the diagram, we require that $\boldsymbol{v}^i_j$ after applying transformation T_i have a 2D projection at $\boldsymbol{p}^i_j$. Formally, we have

$$E_l = \frac{\beta}{N} \sum_{i \in \mathcal{B}} \sum_{j=1}^{N_i} \|P(\mathsf{T}_i \boldsymbol{v}^i_j) - p^i_j\|_2^2 . \quad (8)$$

Here $P(\cdot)$ is the camera's projection operator from 3D to 2D, known from the camera fitting stage (§6.1.1); $\mathcal{B}$ denote the set of furniture components; N is the total number of 2D endpoints of all furniture components; and β is the regularization weight ($\beta = 0.001$ for all our examples).

The cost function sums up the three terms. We minimize it over the transformations T_i of all furniture components. After transformed by T_i, the furniture components can be properly assembled, as shown in Figure 10-b.

6.2.2 Shape Regularization with Geo-semantic Constraints

Only after the 3D shapes are assembled together, we are able to infer geo-semantic relations between parts. Inspired by [Chen et al. 2013], we construct a cost function to optimize the satisfaction of three types of geo-semantic constraints that account for parallel edges, coplanar faces, and similar geometric sizes. Minimizing this cost function produces a new set of T_i that improve the regularity of the reconstructed 3D shapes, an important property of furnitures as man-made objects (Figure 10).

Parallel edges. Consider a pair of 3D parts, B_i and B_j, after applying the transformations computed in §6.2.1. We add a cost function term if an edge $\boldsymbol{a}_i$ on B_i is nearly parallel to another edge $\boldsymbol{a}_j$ on B_j (i.e., the angle inbetween is less than $15°$), namely

$$E_p = \frac{1}{|\mathcal{P}|} \sum_{\{i,j\} \in \mathcal{P}} \|(\mathsf{T}_j \boldsymbol{a}_j) \times (\mathsf{T}_i \boldsymbol{a}_i)\|_2^2 ,$$

where $\mathcal{P}$ is the set of component pairs that have parallel edges.

Coplanar faces. When B_i has a face close to another face on B_j and the angle between both faces are less than $15°$. We use $\boldsymbol{c}_i$ and $\boldsymbol{n}_i$ to denote the face's center point and normal direction in 3D, respectively. and define a coplanarity term,

$$E_f = \frac{1}{|\mathcal{F}|} \sum_{\{i,j\} \in \mathcal{F}} \left(\left\| \frac{\mathsf{T}_j \boldsymbol{c}_j - \mathsf{T}_i \boldsymbol{c}_i}{\|\mathsf{T}_j \boldsymbol{c}_j - \mathsf{T}_i \boldsymbol{c}_i\|_2} \cdot \mathsf{T}_i \boldsymbol{n}_i \right\|_2^2 + \|\mathsf{T}_i \boldsymbol{n}_i - \mathsf{T}_j \boldsymbol{n}_j\|_2^2 \right) .$$

Similar geometric sizes. Additionally, we compute the oriented bounding box of each B_i. Let l_i be one side length of B_i, and l_j is for B_j, we add the following size similarity term if the length difference between l_i and l_j is less than 5%:

$$E_s = \frac{1}{|S|} \sum_{\{i,j\} \in \mathcal{S}} \|\mathsf{T}_j(l_j) - \mathsf{T}_i(l_i)\|^2 .$$

We combine the three terms into an objective function and solve for a new set of T_i initialized with the transformations from §6.2.1. The resultant T_i are close to the initialization, so the assembly constraints are retained. But they regularize the 3D shape reconstruction, as shown in Figure 10.

6.3 Dynamic Assembly Process

With the static shapes of furniture components reconstructed, we now recover the dynamic assembly process described in a single instruction step. To this end, two visual element types are informative and essential to our algorithm.

Connectors. A mechanical connector indicates how two components are joined together. Through optimizing shape transformations T_i in §6.2.1, we have already aligned 3D shapes to their connectors and known which two faces need to touch each other. In this stage, we simply translate the 3D components along the connector's join direction until the two faces touch.

We also recover the process of mount mechanical connectors onto the furniture. Recall that in §6.2.1 we have detected the connection points. Let $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ denote two connection points on two touching faces (Figure 10). We align the connector with the line segment $\boldsymbol{h}_i\boldsymbol{h}_j$ and insert it along the line's direction. We also scale the connector's 3D model until its 2D projection on the image plane matches that in the assembly diagram.

Arrows. In other cases, an arrow is used to indicate an assembly action (Figure 2). To detect the arrow's pointing direction, we start with a predefined arrow shape A_0 and use 2D shape matching [Shao et al. 2011] to compute an affine transformation T_0 that matches A_0 with the arrow shape in the diagram. We then use T_0 to transform A_0's initial direction and obtain the arrow's pointing direction. Next, searching in a local region around the arrow, we find the furniture components that are in proximity to the arrow's head and tail. The furniture component near its head is the assembled component, while one near its tail is the target component. Lastly, we move the assembled component along the arrow's point direction until it touches the target component.

There are also 3D arrows depicted using curved paths (e.g., see step 7 in Figure 2). We distinguish 3D arrows from 2D arrows by checking whether they consist of curved paths or only straight paths. These arrows are used to indicate the change of orientation of a furniture component or the rotation of screws. Since the orientation changes will be identified through the cross-step correspondence later in §7, and since the screws are always inserted along the pointed direction, there is no need to rely on 3D arrows to recover assembly actions. Thus, we ignore them.

7 Cross-Step Correspondence

With the algorithm operating on a single instruction step presented, we move on to process multiple instruction steps. An assembly instruction is organized in an incremental fashion across multiple steps: a later step depicts assembly of new furniture components added to previously assembled parts. Algorithmically, the key task is to recognize previously assembled furniture components in a new instruction step.

Challenges. We aim to address three challenges in this stage: (i) The view angle can change drastically across steps, so existing furniture components may have very different 2D projections on a new step. (ii) More notable is the problem of occlusion: as furniture components and mechanical connectors are progressively assembled, more components become occluded, rendering individual 3D fitting unreliable. (iii) The projective imprecision remains a problem. Even with a precisely known view angle, the projection of previously assembled parts may not fully match their depiction on a new step.

Formally, given a new step of assembly diagram and a group of previously assembled furniture parts $\mathcal{G} = \{G_0, G_1, ..., G_m\}$ in 3D, we detect if any of the parts in $\mathcal{G}$ appears in the new diagram. If so, we need to transform the assembled furniture parts to align with their projection in the new diagram. For this purpose, our algorithm takes the following two stages.

Estimating view angle. Using the parsing algorithm in §5, we detect furniture components in the current assembly diagram. We then group them into connected components based on their proximity in the diagram. Suppose there are n connected furniture components, named as $\mathcal{C} = \{C_0, C_1, ..., C_n\}$. Next, we select 72 view angles by uniformly sampling the surface of a unit sphere. For each view angle, we project furniture parts in $\mathcal{G}$ onto the image plane, and the 2D projections are $\tilde{\mathcal{G}} = \{\tilde{G}_1, ..., \tilde{G}_m\}$. In practice, we implement the sampling and projection in OpenGL to leverage the fast and parallel performance of Graphic hardware. We again use the 2D shape matching algorithm [Shao et al. 2011] to check if a part in $\tilde{\mathcal{G}}$ matches a component in $\mathcal{C}$. We choose this algorithm because of its ability to handle partial matching, a feature essential when C_i is occluded by other parts or connected with new components introduced in the current step. The sampled view angles might not precisely align with the true view angle. Therefore, after establishing a matching between a furniture part $\tilde{G}_i$ and a component C_j, we improve the view angle using the standard 3D-to-2D iterative closest point (ICP) algorithm [Besl and McKay 1992].

When $\tilde{G}_i$ partially matches C_j, we identify the fully matched furniture component in C_j as the previously assembled part G_i, and therefore do not build a new 3D model for that furniture component. In addition, we establish a set of correspondence relations between the 3D endpoints of edges in G_i and their corresponding 2D points in C_j. We denote this set of 3D points as $\mathcal{V}_{ij} = \{\boldsymbol{v}_{ij,1} ... \boldsymbol{v}_{ij,k}\}$ and their corresponding 2D points as $\mathcal{P}_{ij} = \{\boldsymbol{p}_{ij,1} ... \boldsymbol{p}_{ij,k}\}$. We will use this correspondence and notation in the next stage.

Transforming existing shapes. To reuse G_i in the dynamic 3D reconstruction process (as presented in §6) of the current diagram, we need to transform G_i to align with the new furniture components.

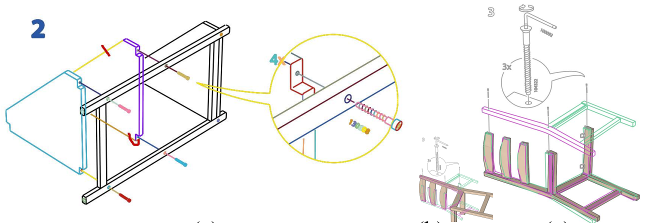

Figure 11: User interactions. **(a)** *The user uses strokes (in red) to add (or remove) paths into (or from) a selected group. Given an incorrect 3D-2D alignment shown in* **(b)***, the user drags the 3D part to the approximately correct location* **(c)***. It is then automatically snapped to align with the 2D drawing (see video).*

Similar to Eq. (8), we estimate T_i by minimizing the cost function,

$$E_t = \frac{1}{|\mathcal{N}|} \sum_{(i,j)\in\mathcal{N}} \sum_{k} \|\mathbf{P}(\mathsf{T}_j \boldsymbol{v}_{ij,k}) - \boldsymbol{p}_{ij,k}\|_2^2 ,$$

where $\mathcal{N}$ denote all pairs of matchings between $\tilde{\mathcal{G}}$ and $\mathcal{C}$. Here we safely ignore the connector and component constraints used in §6.2.1, as they have been enforced when we construct the 3D furniture parts G_i earlier. After solving this optimization problem, we transform the assembled furniture shapes with the computed T_i. The rest of the furniture components are newly introduced in the current instruction step. We therefore follow the algorithm in §6 to construct their 3D shapes and dynamic assembly process.

8 User Interaction

Occasionally, the pipeline presented above can not handle diagrams fully correctly. This is because, like many machine learning algorithms, our parsing algorithm, while having a high accuracy (see Table 4 and the supplemental document), can fail in certain cases. We therefore provide a user interface to allow interactive corrections.

In our pipeline, three stages may require user correction. (i) In the path grouping stage (§5.1), the user can click and select individual paths, and add them into or remove them from a group (Figure 11-a). (ii) In the recognition stage (§5.2), the user can click and select a path group, right-click to pop up a menu, and assign the group a correct visual element type. (iii) When we build cross-step correspondence (§7), an existing 3D model may be incorrectly aligned with a 2D drawing. We allow the user to drag the 3D model to an approximately correct position (Figure 11-b). Then the 3D shape is snapped to align with the drawing automatically using our 3D-to-2D ICP algorithm. The supplemental video illustrates these user interaction stages. As summarized in Table 4 and the supplemental document (for all our examples), our system requires a small amount of user interactions; most instruction steps are automatically processed.

9 Evaluation

Experiment Setup. We extensively evaluated our method using a wide variety of assembly instructions downloaded from two sites, Ikea and Nitori. In total, we have tested our method on 70 furniture assembly instructions. These examples are chosen by checking almost all furniture categories available on the websites, and we test our method over the assembly instructions, in which the furnitures are extruded shapes, since our method is not designed for other types of furnitures such as those made by rods, clothes, and linkages. In the end, our examples span a wide range of furniture types, covering outdoor furniture, baby and children products, bathroom

Example	Shelf	TV Cabinet	Stool	Bedside table	Desk	Underbed	Chair	Footstool	Filing box	Storage box	Bench	Coffee table
#Steps	3	11	8	15	32	9	7	5	8	18	13	8
#Parts	8	11	15	11	23	23	13	5	6	7	11	6

Table 1: Statistics. *The complexity of instructions of 12 representative assembly instructions. Other examples are reported in the supplemental document.*

storage, decoration, kitchen cabinets and appliances, and so on. To avoid copyright issues, these examples presented in the paper were redrawn by our artists.

For the sake of saving space, we include 12 representative examples here, and refer to the supplemental document (and the secondary supplemental video) for all the examples. For the 12 examples, we summarize the complexity of the instructions in Table 1 and the evaluation results in Table 4, including the accuracy of individual stages and the user interaction cost. The statistics of the full set of 70 examples is reported in the supplemental document. Among the 70 examples, the authors modeled only 10 of them, and the remaining 60 examples were modeled by four students in the Master Program of the Computer Science Department. These students could quickly adopt our user interfaces after about 10 minutes of training.

In order to evaluate the accuracy of all the examples, we painstakingly labeled the paths, correct the segmentation, and indicate the type of each visual elements for all of them. The manually labeled data is only used as a ground truth to evaluate the accuracy of different stages of our algorithm. The accuracy in Table 4 is measured by the ratio of correct operations (as indicated by comparing to the ground truth data) to the total operations. We report the number of user interactions needed for segmentation and recognition in a format of "a/b" in Table 4, in which "a" indicates the number of visual elements that require user corrections, and "b" indicates the total number of visual elements detected in that instructions. The same format is used for reporting the number of interactions of the cross-step correspondence stage, for which "a" indicates the number of cross-step correspondence pairs that need user corrections, and "b" is the total number of cross-step correspondence pairs. Finally, to evaluate the quality of the optimizations, the column of "vanishing error" reports the residual of Eq. 6, while the column of "assembly error" reports the residual of $E_a + E_c + E_l$ used in §6.2.

9.1 Validation of Semantic Segmentation

We start by evaluating our semantic segmentation algorithm. To create ground truth for the validation, we manually labeled 172 pages from assembly instructions, separated from those used in our examples, and group the graphic primitives into semantic components. In total, our testing data contain 611 furniture parts, 796 mechanical connectors, 271 arrows, 1652 numbers, 367 highlights and 201 other components (e.g., screwdriver and 2D primitives in highlights). Every assembly step also produces a graph (recall Section 5.1), for which we also manually label the connections of graph edges, and select 420 "grouped" edges and 420 "ungrouped" edges for the evaluation of segmentation. In a different experiment, these labeled data are used to train our algorithm and generate the 70 examples. There is no overlap between the labeled data and the examples.

Labeling accuracy of graph edges. We use cross-validation to evaluate the labeling accuracy of graph edges of our algorithm (§5.1). Specifically, we randomly split the labeled data into 10 validation groups. Given a group, we train a random regression forest using the data from the remaining 9 groups. This process is repeated for every single group. We use the average accuracy to measure the performance of our algorithm. In these tests, our algorithm is able to achieve a 99% training accuracy and a 83% testing accuracy on average.

	Training Accuracy	Testing Accuracy
HO (local)	0.955	0.771
SIFT (local)	0.990	0.783
HO (full)	0.991	**0.836**
SIFT (full)	0.993	0.802

Table 2: *Evaluation of segmentation using HO and SIFT feature.*

Comparison of features. In Table 2, we compare the use of different features. "HO" indicates the use of our HO features, while "SIFT" indicates the use of the SIFT features [Lowe 1999], which have been commonly used in pixel-image segmentation. In addition, "local" uses only local features, while "full" includes the context features, as introduced in §5.1. The experiments show that while SIFT feature leads to higher training accuracy, our algorithm including both the local and context features produces the highest testing accuracy, and thus is indeed suitable for segmenting 2D assembly diagrams. Moreover, Table 2 also shows that the use of context features improves the performance.

Remark. We notice that the average accuracy of HO is not significantly higher than the use of SIFT. However, HO is more robust if the testing example has different drawing styles in its vector graphics depiction. This is desired, as we found that instruction diagrams, depicted with vector graphic paths, may use different line widths and drawing sizes in many cases. HO features are consistent when the line width and drawing size vary, but SIFT features would change in those cases, because SIFT features are computed in a local region with a fixed resolution (we use 10pt in our experiments for the best SIFT performance).

Recognition accuracy. We also use cross-validation to evaluate the recognition accuracy on the labeled data. Again, we randomly split these data into 10 validation groups and perform a cross-validation similar to the aforementioned tests. Table 3 summarizes the recognition accuracy for each visual element category, showing that our algorithm can reach over 90% recognition accuracy for all the 5 categories. Here we measure the recognition accuracy after the grouping stage (§5.1) with user correction (if needed), because the recognition stage can proceed only when the grouping is fully correct.

	Part	Connector	Number	Arrow	Highlight
Train	0.990	0.993	0.996	0.981	0.976
Test	0.920	0.938	0.988	0.901	0.965

Table 3: *Evaluation of recognition using GALIF features.*

9.2 Validation of 3D fitting

Vanishing point optimization. As illustrated in Figure 8, our optimization of vanishing point can effectively regularize the 3D fitted shapes. This is important as many 2D assembly diagrams have imprecise perspective projections. The average regularization error for individual examples are listed in Table 4 and the supplemental document, and the total average error is $6.0\mathrm{e}^{-4}$.

Assembly constraints. Reconstructing 3D shapes of detected furniture components individually can not guarantee that they fit together. As shown in Figure 10, incorporating the assembly constraints (§6.2.1) and geo-semantic constraints (§6.2.2) improve the

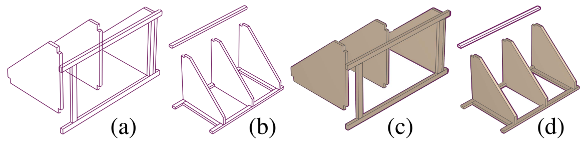

Figure 12: *The view angle changes largely between two steps* **(a-b)**. *Using 3D shapes reconstructed in a previous step* **(c)**, *the correspondences between the two steps are built (§7)* **(d)**.

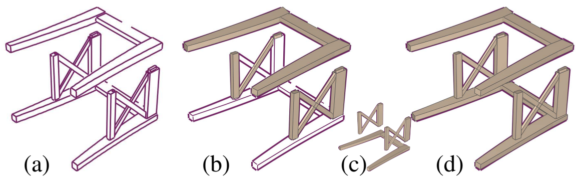

Figure 13: *Provided a 2D drawing of a bench with severe occlusions* **(a)**, *we can only recover unblocked 3D shapes if starting from scratch* **(b)**. *With the existing shapes from previous steps* **(c)**, *a complete 3D shape of the bench can be recovered* **(d)**.

furniture alignment significantly. Again, the average 3D fitting error of this stage (§6.2) for individual examples are summarized in Table 4 and the supplemental document. The total average error is 8.0e^{-4}.

9.3 Validation of Cross-Step Correspondence

We evaluate the accuracy of our cross-step correspondence algorithm using the commonly-adopted metric of Intersection Over Union (IOU), which measures the 3D-to-2D matching score with respect to the manually labeled ground truth. Consider a connected furniture component $\mathcal{C}$. The IOU score is defined as $N_{pos}/(N_C + N_{neg})$, where N_C is the total number of paths of C, N_{pos} is the number of correctly matched paths of C, and N_{neg} is the number of incorrectly matched paths. The average matching score for each example is given in Table 4 and the supplemental table, and the total average score is 0.868.

Large viewpoint change. Our method can reliably handle large change of view angles across two steps, as shown in Figure 12 and correctly match the assembled 3D furniture parts with the drawing in a new assembly diagram.

Heavy occlusions. Our method is also robust to handle severe occlusions by exploiting cross-step correspondence. One example is illustrated in Figure 13.

Improvement of segmentation accuracy. Figure 14 shows an example where cross-step correspondence also helps to improve segmentation. This is also confirmed in Table 4 and the supplemental table: the segmentation accuracy for our final examples is generally higher than the accuracy evaluated through cross-validation in §9.1, where no cross-step correspondence is exploited. In addition, we quantitatively compared the segmentation accuracy with and without the use of cross-step correspondence in our 12 representative examples and summarize the results in Figure 15. In almost all examples, segmentation accuracy is improved with the use of cross-step correspondence.

10 Applications

We now present three applications of our system. We first apply our method to generate 3D animated furniture assembly processes. Next, with 3D reconstructed shapes along with their semantic constraints, we semantically edit furniture parts and customize with

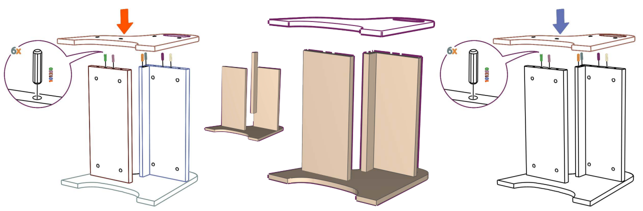

Figure 14: *Two boards are incorrectly labeled "grouped" (left, marked in blue). By exploiting the cross-step correspondence (middle) from previous step, we avoid this confusion and only need to segment paths that can not be matched with existing 3D shapes. On the right, the components with black outlines are those transformed from previous steps.*

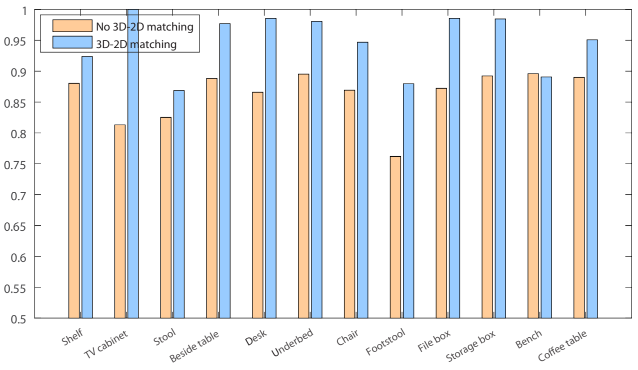

Figure 15: *Comparison of the segmentation accuracy with/without the use of cross-step correspondence in our examples.*

user-specified patterns. Lastly, we physically fabricate the furnitures using 3D digital fabrication and woodworking.

Animated furniture assembly process. We generated 3D assembly animations for 9 representative examples, showing in the supplemental videos (including the main video and the additional video). Among the examples, the number of assembly steps varies from 3 steps (i.e., the storage box) to 32 steps (i.e., the desk). The number of furniture parts is between 5 and 23.

Here we highlight a few examples, which we found is hard to follow by looking at the 2D assembly instructions only. The desk example undergoes significant view angle changes, and the furniture part being assembled switches from the drawer to the desk body. With our 3D animated assembly processes, the view point is smoothly interpolated across steps. When the animation is playing back, our system also displays a completed furniture model, along with the currently assembled furniture parts. By highlighting the currently operated part in the completed furniture model, we can also convey the user a grand picture at every step to keep the user on track. At any point during the playback, the user can freely pan the camera, zoom in and out.

We note that the 3D animation can not be simply produced by interpolating between steps, for two main reasons: First, the 2D projection of the same 3D furniture part can vary drastically in different steps. Second, over a sequence of instruction steps, some existing parts in an earlier step can disappear in a later step, sometimes making the interpolation simply impossible.

Physical fabrication. Given the reconstructed complete 3D shapes, the user is able to fabricate their own furnitures. We demonstrate with two fabrication methods. The first is using 3D printers to fabricate furniture parts (see Figure 17 for a few examples). The second is using traditional woodworking by cutting wooden boards

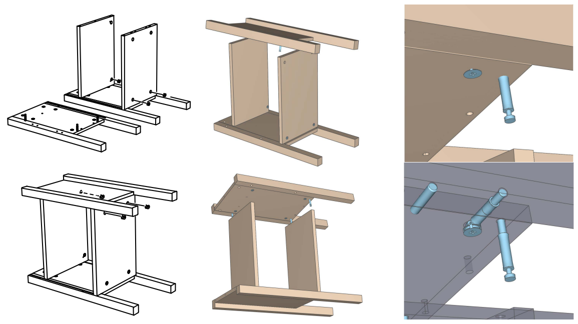
Figure 16: *Our animation allows the user to view the intermediate assembly process between static steps (left) from the view point of drawing (middle up). The user can change the view point to see more details (middle bottom), and zoom in to check how the parts are assembled with connectors (right).*

to furniture parts, and assembling these parts using mechanical connectors purchased at the market. We fabricate both the desk and the bedside table, and generated a physical and functional furnitures (Figure 1 and Figure 18).

Semantic-aware furniture editing. The reconstructed dynamic assembly process not only produces correct 3D furniture parts that can be assembled but recovers furniture semantics such as relative movements and junctions between parts when the furniture is functioning. Exploiting these semantics allows semantic-aware furniture editing, for which we demonstrate two types of editing. The first is for posing the furnitures. We use motion arrows [Shao et al. 2013] (pink for rotation and blue for sliding) to indicate the furniture's degrees of freedom (see Figure 19-b). The user is able to drag the arrows to repose the furniture. The second edit is semantic-aware shape change [Zheng et al. 2012; Chen et al. 2013]: the user initiates an edit on a selected furniture part, and the algorithm automatically propagates the edit to other parts that are semantically related to the selected part to satisfy e.g., coplanar, same-size and symmetric relations (see Figure 19).

Furniture customization. Once obtaining a complete 3D furniture model, the user can further decorate the furniture with personalized patterns. In our example, we add flower patterns to the surface of a chair and a footstool (see Figure 17). We demonstrate this customization by physically fabricating the results.

11 Limitations and Future Work

We have demonstrated the robustness of our method with a wide range of furniture assembly instructions. While producing promising results, our method has limitations.

First, our system can not process instructions involving furniture shapes that are not vertically cut from boards (e.g. sofas, bins and bags, and linkages) (Figure 20-a), because we consider only extruded shapes whose extrusion edges are detected based on parallel lines and their lengths. We are interested in extending our method

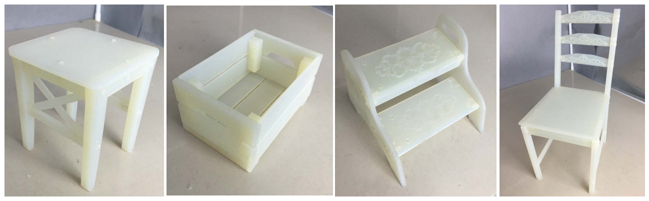
Figure 17: *3D printed furniture: the right two were customized with user-selected patterns (see video).*

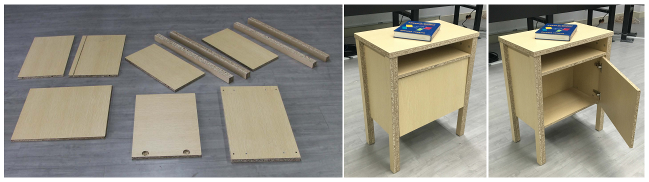
Figure 18: *A functional bedside table reconstructed and made with woodworking.*

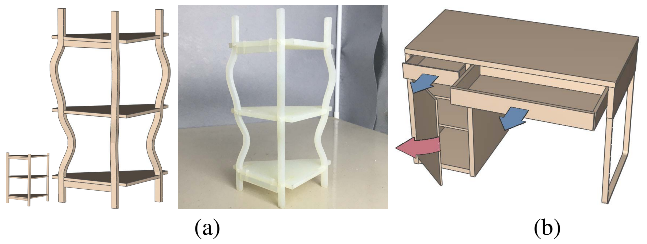
Figure 19: Semantic-aware furniture editing. (a) *A shelf is semantically edited and then 3D printed.* **(b)** *The desk can be reposed using the semantic arrow handles.*

to handle other types of furnitures. For instance, Chen et al. [2013] describes a method to model cylindrical structures from a single image, which can be incorporated in our system.

Our pipeline aims to handle a wide range of furniture assembly instructions from different manufacturers and brands. Yet, the visualization style of the instructions across different brands may vary in a subtle way. For instance, some have a list of individual furniture parts at the beginning, and some (like those from Ikea) do not. We choose not to use the information that is inconsistent across different instructions. In some instructions from Nitori, the model numbers of some mechanical connectors are missing, so we can not retrieve the connector models (Figure 20-c). In our implementation, we choose to ignore those connectors or rely on the user manually specifying the model numbers.

So far, our method neglects 3D arrows and highlights in assembly diagrams, 3D arrows are meant to indicate rotations of certain furniture components. While we currently infer furniture rotations based on the detected cross-step correspondence among furniture parts, it would be interesting to further exploit the 3D arrows and possibly improve the accuracy of the correspondence detection. We are also interested in further exploiting the highlights, with the hope of reconstructing the models of connectors whose model numbers are missing.

We assume that an occluded furniture component in an instruction step has an unoccluded counterpart in previous steps. However, we found that in a few cases this assumption breaks (Figure 20-b). One possible solution is to automate a drawing completion based on semantic prior knowledge and symmetry detection.

Currently, our pipeline needs user interactions, mainly due to the inability of our learning algorithm based on HO features for segmenting and grouping vector graphic primitives fully correctly. Better features and improved recognition techniques are certainly worth exploring.

Currently, we collect our training data by manually labeling a small set of instructions. This is a laborious process. A better approach of generating training data can benefit from automatically generated furniture instructions (e.g., using [Agrawala et al. 2003]), by using the resulting instructions for training.

Lastly, we realize that the method can be abused to counterfeit furniture designs, giving rise to legal challenges. In history, the copyright

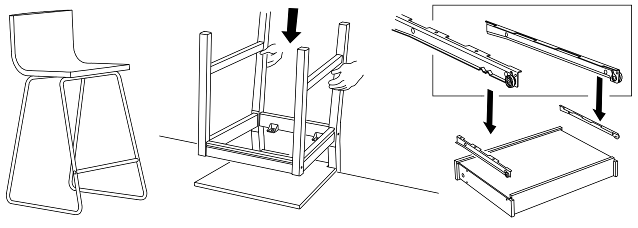

Figure 20: Failure cases. **(a)** *Non-extrusion shapes are not supported.* **(b)** *Our method would fail if an occluded 3D part such as the chair seat has not appeared in earlier steps.* **(c)** *If there is no model number for a mechanical connector, we have to ignore that connector or rely on the user specifing its model number.*

always became an issue when new techniques emerged (recalling the digitalization of music and books). Addressing the copyright issue of furniture design opens a future direction.

Acknowledgements

We thank the anonymous reviewers for their feedback. This work was supported in part by the NSF of China (No. 61272305 and No. 61402402), the National Program for Special Support of Eminent Professionals of China, the National Science Foundation (CAREER-1453101) and generous gifts from Lenovo, Intel and Adobe.

References

AGRAWALA, M., PHAN, D., HEISER, J., HAYMAKER, J., KLINGNER, J., HANRAHAN, P., AND TVERSKY, B. 2003. Designing effective step-by-step assembly instructions. *ACM Trans. Graph. 22*, 3 (July), 828–837.

BAE, S.-H., BALAKRISHNAN, R., AND SINGH, K. 2008. Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of UIST*, ACM, UIST '08, 151–160.

BERTHOUZOZ, F., GARG, A., KAUFMAN, D. M., GRINSPUN, E., AND AGRAWALA, M. 2013. Parsing sewing patterns into 3d garments. *ACM Trans. Graph. 32*, 4 (July), 85:1–85:12.

BESL, P. J., AND MCKAY, N. D. 1992. A method for registration of 3-d shapes. *IEEE TPAMI 14*, 2, 239–256.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *IEEE TPAMI 23*, 11, 1222–1239.

CAO, Y., JU, T., FU, Z., AND HU, S. 2014. Interactive image-guided modeling of extruded shapes. *Comput. Graph. Forum 33*, 7, 101–110.

CHEN, X., KANG, S. B., XU, Y.-Q., DORSEY, J., AND SHUM, H.-Y. 2008. Sketching reality: Realistic interpretation of architectural designs. *ACM Trans. Graph. 27*, 2 (May), 11:1–11:15.

CHEN, T., ZHU, Z., SHAMIR, A., HU, S.-M., AND COHEN-OR, D. 2013. 3sweep: Extracting editable objects from a single photo. *ACM Trans. Graph. 32*, 6 (Nov.), 195:1–195:10.

DE PAOLI, C., AND SINGH, K. 2015. Secondskin: Sketch-based construction of layered 3d models. *ACM Trans. Graph. 34*, 4 (July), 126:1–126:10.

EITZ, M., HAYS, J., AND ALEXA, M. 2012. How do humans sketch objects? *ACM Trans. Graph. 31*, 4 (July), 44:1–44:10.

EITZ, M., RICHTER, R., BOUBEKEUR, T., HILDEBRAND, K., AND ALEXA, M. 2012. Sketch-based shape retrieval. *ACM Trans. Graph. 31*, 4 (July), 31:1–31:10.

FANELLI, G., GALL, J., AND GOOL, L. J. V. 2011. Real time head pose estimation with random regression forests. In *CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, 617–624.

FU, H., ZHOU, S., LIU, L., AND MITRA, N. J. 2011. Animated construction of line drawings. *ACM Trans. Graph. 30*, 6 (Dec.), 133:1–133:10.

FUNKHOUSER, T., MIN, P., KAZHDAN, M., CHEN, J., HALDERMAN, A., DOBKIN, D., AND JACOBS, D. 2003. A search engine for 3d models. *ACM Trans. Graph. 22*, 1 (Jan.), 83–105.

GENNARI, L., KARA, L. B., STAHOVICH, T. F., AND SHIMADA, K. 2005. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics 29*, 4, 547–562.

GINGOLD, Y., IGARASHI, T., AND ZORIN, D. 2009. Structured annotations for 2d-to-3d modeling. *ACM Trans. Graph. 28*, 5 (Dec.), 148:1–148:9.

GUPTA, A., FOX, D., CURLESS, B., AND COHEN, M. 2012. Duplotrack: A real-time system for authoring and guiding duplo block assembly. In *Proceedings of UIST '12*, 389–402.

HARALICK, R. M., AND QUEENEY, D. 1982. Understanding engineering drawings. *Computer Graphics and Image Processing 19*, 1, 90.

HARLTEY, A., AND ZISSERMAN, A. 2006. *Multiple view geometry in computer vision (2. ed.).* Cambridge University Press.

HEISER, J., PHAN, D., AGRAWALA, M., TVERSKY, B., AND HANRAHAN, P. 2004. Identification and validation of cognitive design principles for automated generation of assembly instructions. In *Proceedings of the working conference on Advanced Visual Interfaces*, ACM, 311–319.

HUANG, Z., FU, H., AND LAU, R. W. H. 2014. Data-driven segmentation and labeling of freehand sketches. *ACM Trans. Graph. 33*, 6 (Nov.), 175:1–175:10.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. In *Proceedings of ACM SIGGRAPH '99*, 409–416.

JR., J. J. L., AND ZELEZNIK, R. C. 2004. Mathpad^2: a system for the creation and exploration of mathematical sketches. *ACM Trans. Graph. 23*, 3, 432–440.

KARPENKO, O. A., AND HUGHES, J. F. 2006. Smoothsketch: 3d free-form shapes from complex sketches. *ACM Trans. Graph. 25*, 3 (July), 589–598.

KOO, B., LI, W., YAO, J., AGRAWALA, M., AND MITRA, N. J. 2014. Creating works-like prototypes of mechanical objects. *ACM Trans. Graph. 33*, 6 (Nov.), 217:1–217:9.

LAU, M., OHGAWARA, A., MITANI, J., AND IGARASHI, T. 2011. Converting 3d furniture models to fabricatable parts and connectors. *ACM Trans. Graph. 30*, 4 (July), 85:1–85:6.

LAVIOLA, J., DAVIS, R., AND IGARASHI, T. 2006. An introduction to sketch-based interfaces. *ACM SIGGRAPH Course Notes*.

LI, W., AGRAWALA, M., AND SALESIN, D. 2004. Interactive image-based exploded view diagrams. In *Proceedings of Graphics Interface 2004*, GI '04, 203–212.

Examples	Seg.Acc	Seg.Inter	Rec.Acc	Rec.Inter	Match.Acc	Match.Inter	Time(sec.)	Vanish. err.	Assembly. err.
Shelf	0.923	1 / 60	0.946	1 / 60	0.915	0/2	10.6	$2.0e^{-4}$	$7.0e^{-4}$
TV cabinet	1.000	0 / 452	0.986	6 / 452	0.958	1/9	42.0	$1.0e^{-3}$	$5.0e^{-4}$
Stool	0.868	3 / 124	0.895	14 / 124	0.937	2/7	111.2	$2.4e^{-3}$	$1.2e^{-3}$
Beside table	0.977	1 / 257	0.968	6 / 257	0.843	2/19	75.7	$2.9e^{-3}$	$8.0e^{-4}$
Desk	0.985	1 / 625	0.947	28 / 625	0.911	1/36	163.0	$1.0e^{-4}$	$5.0e^{-4}$
Underbed	0.980	4 / 216	0.939	19 / 216	0.839	0/17	88.5	$1.9e^{-3}$	$8.0e^{-4}$
Chair	0.946	6 / 111	0.855	15 / 111	0.779	2/5	90.0	$7.0e^{-4}$	$1.6e^{-3}$
Footstool	0.879	4 / 104	0.990	1 / 104	0.936	0/4	41.7	$5.0e^{-4}$	$1.0e^{-3}$
Filing box	0.985	0 / 48	1.000	1 / 48	0.902	2/11	38.8	$2.0e^{-4}$	$7.0e^{-4}$
Storage box	0.984	0 / 646	0.982	9 / 646	0.949	4/23	82.5	$2.0e^{-3}$	$9.0e^{-4}$
Bench	0.890	1 / 251	0.904	25 / 251	0.881	0/16	102.5	$3.0e^{-4}$	$1.1e^{-3}$
Coffee table	0.950	1 / 153	0.938	5 / 153	0.841	1/7	33.7	$3.0e^{-4}$	$5.0e^{-4}$

Table 4: Statistics *of the representative examples. From left to right, the columns report segmentation accuracy, # user interactions needed for segmentation, recognition accuracy, # user interactions needed for recognition, cross-step matching accuracy, # user interactions needed to correct the cross-step correspondence, the total processing time, the residual for vanish point optimization, and the residual fo rassembly optimization (§6.2).*

LI, H., HU, R., ALHASHIM, I., AND ZHANG, H. 2015. Foldabilizing furniture. *ACM Trans. Graph. 34*, 4 (July), 90:1–90:12.

LOWE, D. G. 1999. Object recognition from local scale-invariant features. In *ICCV*, 1150–1157.

MENA, J. B. 2003. State of the art on automatic road extraction for GIS update: a novel classification. *Pattern Recognition Letters 24*, 16, 3037–3058.

MITRA, N. J., YANG, Y.-L., YAN, D.-M., LI, W., AND AGRAWALA, M. 2010. Illustrating how mechanical assemblies work. *ACM Trans. Graph. 29*, 4, 58.

MOHR, P., KERBL, B., DONOSER, M., SCHMALSTIEG, D., AND KALKOFEN, D. 2015. Retargeting technical documentation to augmented reality. In *Proceedings of CHI '15*, ACM, New York, NY, USA, 3337–3346.

OUYANG, T. Y., AND DAVIS, R. 2011. Chemink: a natural real-time recognition system for chemical drawings. In *Proceedings of IUI, 2011*, 267–276.

RONG, Y., ZHENG, Y., SHAO, T., YANG, Y., AND ZHOU, K. 2016. An interactive approach for functional prototype recovery from a single rgbd image. *Computational Visual Media 2*, 1, 87–96.

SAUL, G., LAU, M., MITANI, J., AND IGARASHI, T. 2011. SketchChair: An All-in-one Chair Design System for End Users. In *Proceedings of TEI*, ACM, TEI '11.

SCHMIDT, R., KHAN, A., SINGH, K., AND KURTENBACH, G. 2009. Analytic drawing of 3d scaffolds. *ACM Trans. Graph. 28*, 5 (Dec.), 149:1–149:10.

SCHULZ, A., SHAMIR, A., LEVIN, D. I. W., SITTHI-AMORN, P., AND MATUSIK, W. 2014. Design and fabrication by example. *ACM Trans. Graph. 33*, 4 (July), 62:1–62:11.

SHAO, T., XU, W., YIN, K., WANG, J., ZHOU, K., AND GUO, B. 2011. Discriminative sketch-based 3d model retrieval via robust shape matching. *Comput. Graph. Forum 30*, 7, 2011–2020.

SHAO, T., LI, W., ZHOU, K., XU, W., GUO, B., AND MITRA, N. J. 2013. Interpreting concept sketches. *ACM Trans. Graph. 32*, 4 (July), 56:1–56:10.

SHOTTON, J., JOHNSON, M., AND CIPOLLA, R. 2008. Semantic texton forests for image categorization and segmentation. In *CVPR 2008, Anchorage, Alaska, USA*.

SHTOF, A., AGATHOS, A., GINGOLD, Y. I., SHAMIR, A., AND COHEN-OR, D. 2013. Geosemantic snapping for sketch-based modeling. *Comput. Graph. Forum 32*, 2, 245–253.

SUN, Z., WANG, C., ZHANG, L., AND ZHANG, L. 2012. Free hand-drawn sketch segmentation. In *ECCV 2012*, 626–639.

TOMBRE, K. 1998. *Graphics Recognition Algorithms and Systems: Second International Workshop, GREC' 97 Nancy, France, August 22–23, 1997 Selected Papers*. ch. Analysis of engineering drawings: State of the art and challenges, 257–264.

TVERSKY, B., ZACKS, J., LEE, P., AND HEISER, J. 2000. Lines, blobs, crosses and arrows: Diagrammatic communication with schematic figures. In *Theory and application of diagrams*. Springer, 221–230.

TVERSKY, B., Y, J. B. M., AND BETRANCOURT, M. 2002. Animation: Can it facilitate. *International Journal of Human-Computer Studies 57*, 247–262.

UMETANI, N., IGARASHI, T., AND MITRA, N. J. 2012. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph. 31*, 4 (July), 86:1–86:11.

XU, K., CHEN, K., FU, H., SUN, W.-L., AND HU, S.-M. 2013. Sketch2scene: Sketch-based co-retrieval and co-placement of 3d models. *ACM Trans. Graph. 32*, 4 (July), 123:1–123:15.

XU, B., CHANG, W., SHEFFER, A., BOUSSEAU, A., MCCRAE, J., AND SINGH, K. 2014. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Trans. Graph. 33*, 4 (July), 131:1–131:13.

ZAUNER, J., HALLER, M., BRANDL, A., AND HARTMANN, W. 2003. Authoring of a mixed reality furniture assembly instructor. In *ACM SIGGRAPH 2003 Sketches & Applications*.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 2006. Sketch: An interface for sketching 3d scenes. In *ACM SIGGRAPH 2006 Courses*, ACM, New York, NY, USA, SIGGRAPH '06.

ZHENG, Y., CHEN, X., CHENG, M.-M., ZHOU, K., HU, S.-M., AND MITRA, N. J. 2012. Interactive images: Cuboid proxies for smart image manipulation. *ACM Trans. Graph. 31*, 4 (July).