Robotica (2023), **41**, pp. 48–73
doi:10.1017/S0263574722000443

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Fabrication-aware design for furniture with planar pieces

Wenzhong Yan[1,*], Dawei Zhao[2] and Ankur Mehta[3]

[1]Mechanical and Aerospace Engineering Department, UCLA, Los Angeles, CA, USA, [2]Computer Science Department, UCLA, Los Angeles, CA, USA, and [3]Electrical and Computer Engineering Department, UCLA, Los Angels, CA, USA
*Corresponding author. E-mail: wzyan24@g.ucla.edu

Received: 15 January 2020; **Revised:** 4 January 2022; **Accepted:** 2 March 2022; **First published online:** 11 April 2022

Keywords: flat-pack furniture, furniture design, fabrication-aware design, parameterized abstraction, hierarchical composition

Abstract

We propose a computational design tool to enable casual end-users to easily design, fabricate, and assemble flat-pack furniture with guaranteed manufacturability. Using our system, users select parameterized components from a library and constrain their dimensions. Then they abstractly specify connections among components to define the furniture. Once fabrication specifications (e.g., materials) designated, the mechanical implementation of the furniture is automatically handled by leveraging encoded domain expertise. Afterwards, the system outputs three-dimensional models for visualization and mechanical drawings for fabrication. We demonstrate the validity of our approach by designing, fabricating, and assembling a variety of flat-pack (scaled) furniture on demand.

1. Introduction

Three-dimensional (3D) objects built from planar pieces have drawn extensive attention and been widely applied to owe to their properties, including high strength-to-weight ratio [1], rapid design, and prototyping [2], low cost [3], compact storage, and transport [4]. Recently, digital fabrication techniques have greatly increased the ability of casual end-users to create certain physical objects by reducing necessary design and manufacturing investments. However, the creation of functional furniture is still limited to domain experts due to requirements of in-depth engineering understanding for design, skilled carpentry expertise for fabrication and assembly, and material resources to facilitate the whole process. To bring digital fabrication to this space, we have developed a computational design pipeline enabling casual end-users to easily handle the whole creation process of flat-pack furniture, from design, through fabrication, to assembly.

In our system, the design process is abstracted and parameterized, which allows users to easily design their furniture models in a function-based manner without worrying about the low-level engineering implementation. Besides a conventional incremental method, we also harness a hierarchical composition scheme, which further facilitates and accelerates the design process by providing a recursive approach to build complex models from relatively simple designs through combining functions. Moreover, an intersection autodetection algorithm is employed to automatically identify connections that are not specified but necessary for assembly, and insert planar joints to finalize the designs. In other words, users merely need to specify a minimal number of connections that define the spatial structures of their furniture models; our system will automatically detect other necessary connections and insert specific joints accordingly to generate fabricable embodiment of user-defined models. This algorithm releases users from the tedious connection (and joint) specification process, increasing the flexibility of design process by enabling users to freely build their furniture without being concerned with the order of design. In addition, the incorporation of embedded planar joints within our abstracted and parameterized design scheme greatly reduces the complexity of resulting models and increases the feasibility of assembly for

casual end-users. Combining all these features, our system enables casual end-users to easily and intuitively design, fabricate, and assemble flat-pack furniture with guaranteed manufacturability. Using our system, users mainly need three steps to create their desired furniture. First, designers select components (or use composed designs as components) from our library. Then they abstractly define dimensions of selected components and specify necessary connections between components to build furniture models. Our system will automatically handle the detailed fabrication-aware processing to output files for 3D visualization and 2D fabrication. Finally, users assemble their furniture with ease thanks to the embedded planar joint design. The creation process is simple and intuitive, allowing rapid and easy design of sophisticated furniture for casual end-users. By adopting open-source software, inexpensive raw materials, and generalizable fabrication processes, our system expands the accessibility of personalized design to a broader set of nonexpert users. In summary, this work specifically simplifies design, fabrication, and assembly of the creation process, which is then followed by manual part assembly. We address questions of assemblability (i.e., sequencing the motions required to join the constituent pieces) via low-cost substitutes (e.g., copy paper or acrylic sheet) to build prototypes to validate their feasibility. The field of assembly sequence planning (ASP) could potentially be used to solve this problem more efficiently thanks to the widely available similar research that has been done for products of structure engineering [5], mechanical parts [6, 7], concrete buildings [8], and furniture [9, 10].

In this paper, we present the following specific contributions:

- a computational design pipeline that allows casual end-users to easily generate customizable, manufacturable, easy-to-assemble flat-pack furniture designs;
- an extensible framework that enables users to create furniture designs of arbitrary complexity by hierarchically composing preexisted designs in a function-based manner;
- an algorithm that greatly increases the feasibility and flexibility of design process through automatically detecting and inserting necessary joints (connections) for user-defined furniture models; and
- a representative variety of (scaled) furniture designed and fabricated using the proposed system.

2. Related work

This paper is inspired by the architecture of previous work [11, 12], which exclusively targets origami-inspired structures—3D geometries folded from stock sheets of negligible thickness—to create a system that translated structural specifications into a fully functional printable robot. Here, we focus on flat-pack furniture creation with thick sheet materials instead of origami-inspired structures, which brings in new challenges to design. We also draw upon other academia in the broad categories of modeling by example, fabrication-aware design and personal fabrication.

2.1. Modeling by example

Shape collections have been widely used to allow data-driven geometric modeling [13]. Modeling by example [14, 15, 16] enables the users to customize their own models by manipulating existing templates from a large database built by domain experts. More recent work uses recombination of model parts to expand the databases [17]. Data-driven suggestions can be used to provide recommendations for designs [18].

Perhaps the closest work to this project in terms of the desired goals comes from Schulz et al. [19], in which models—including furniture—could be physically realized via modeling and fabrication by example. From an expert-created database of fabricable templates of finished designs, users could manipulate parameterized models with automatically positioning, alignment, and composition. This "model by example" process thus allows casual end-users to explore a predefined design space bounded by example designs created by domain experts. However, this "model by example" approach

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

requires extensive efforts from domain experts to build a representative library of the targeted models. In our work, we propose a computational design pipeline for flat-pack furniture. Users are allowed to freely design their desired models with their manufacturability guaranteed.

Our system needs initial seed designs instantiated by experts, but they need not be any more than simple polygons (e.g., triangles and rectangles). From there, other desired planar and nonplanar geometries can be easily composed by casual users from the seed polygons using our hierarchical composition strategy (see Section 4.2). Expert users can further seed more complex assemblies—all the way up to complete furniture designs—to further simplify the design load on the casual user. The main difference from Schulz's work comes from remixing components within that library, which can be done without any expert knowledge in our system; those new designs further extend the library, allowing for ever-growing complexity of designs without expert design.

2.2. Fabrication-aware design

Manufacturability of resulting models has long been a concern in the computer graphics community and attracted increasing interest recently [20]. Fabrication-aware design is proposed to guarantee generating fabricable models with fabrication specifications. It is based on digital parameterization of the building models and then implemented by considering the fabrication specifications through built-in algorithms. These systems with fabrication-aware design aimed at empowering novice users without desired skills to develop real-world designs.

Recently, an interactive system SketchChair [21] was proposed to assist in designing chair models that can easily be fabricated and assembled. This system automatically generates a set of planar pieces that can be intersected along slots to form a 3D realization of a designed chair model. Inspired by the same idea, Chen et al. [22] and Hildebrand et al. [23] attempt to convert 3D structures into a set of simplified and fabricable planar polygons connected by interlocking planar pieces. These ideas are developed by building an interactive system where users can have access to real-time feedback by incorporating structure optimization and analysis [10]. Most of that work employs a planar interlock mechanism to implicitly render the design due to its easy manufacturability and assemblability. This interlock mechanism constrains the achievable design space of furniture. Though this method can generate arbitrary solid geometries, it requires dense interlockers which may consume massive materials and cause material waste [24]. Recently, automated architecture proposed a similar modular strategy to create architecture by using prefabricated plywood building blocks, which can be assembled into a home, office, or coworking space [25]. However, these plywood building blocks require extensive engineering and complicated installation instruction, which limits it within domain experts. Similarly, Groenewolt et al. have presented several methods to enable the computational design of large-scale architecture [26]. Nonetheless, the fabrication and assembly of these components require domain experts.

Our proposed system makes use of an extensible collection of planar joints to connect different planar elements to generate furniture designs. Similar to the interlocking slots, these joints are rapidly fabricable due to their 2D geometries. A variety of furniture designs have been enabled in this paper. In addition, the design space is potentially extendable thanks to the abstract design scheme. Moreover, other planar joints can be incorporated into our system to enable users to create more different types of furniture.

2.3. Personal fabrication and assembly

Based on the continued development of democratized tools [27, 28, 29, 30], it is expected that casual end-users will play an important role in designing and creating their own products in the future. Several researchers have recently introduced systems for personal fabrication. Spatial Sketch [31] allows users to sketch in a 3D real-world space. This system realizes the personal fabrication by converting the original 3D sketch into a set of planar slices realized using a laser cutter and assembled into solid objects. Similarly, users can create customized plush toys [32] and chairs [21]. These systems convert designs from 3D geometries to a series of planar pieces to simplify fabrication. Postprocessing assembly of

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

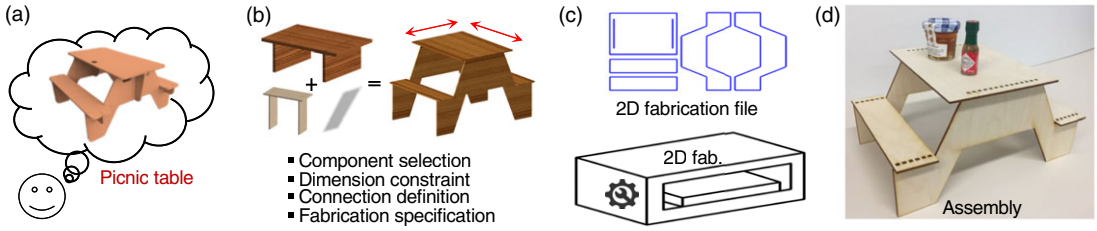

Figure 1. *Workflow for making furniture. We use a picnic table as an example. (a) Conception of designs; (b) Design in our system. Users select components (or designs) from our library, constrains their dimensions, define connections of selected components (or designs), and input fabrication specifications (e.g., materials and corresponding thicknesses); (c) 2D fabrication. The output 2D fabrication file is patterned on planar materials (e.g., plywood, 3 mm) by 2D fabrication machinery (e.g., laser cutter); (d) Assembly. Furniture is built with easy-to-assemble joints through interference fit.*

varying complexity is needed to complete the manufacture. Our work similarly approaches personal fabrication through the use of planar joints which are instinctively easy to assemble, minimizing the need for careful positioning or hardware-based attachments.

3. System overview

In this section, we will overview our system by outlining the design workflow and discussing the design space enabled by our computational method.

3.1. Design workflow

As shown in Fig. 1, our system assists inexpert end-users in handling the whole creation process of flat-pack furniture, from design, through fabrication, to assembly. More specifically, with a furniture model in mind, users can follow these four sequential steps (Fig. 1(b)) to achieve their designs: (1) components selection; (2) parameter constraint; (3) connection definition; and (4) fabrication specification. To demonstrate the process, we use a very simple model (two rectangles connected perpendicularly along an edge—perhaps serving as a bookend—as shown in Fig. 2) as an example with its script-based interface (see Fig. 3). In the first step, casual end-users select two predefined parameterized rectangle components from an existing library (Fig. 2(a), and lines 8–10 in Fig. 3) to match their conception of the designs (experts can create their own components, see Section 4.1.1). Then, they constrain the selected components' geometries, that is, widths and lengths (Fig. 2(b), and lines 16–21 in Fig. 3). In the third step, users specify the design by defining the connections with connected edges and angle (Fig. 2(c), and lines 23–27 in Fig. 3). In the last step, given a description of available fabrication specifications (e.g., tools, materials), our system will automatically produce manufacturable specifications that capture dimensioned geometries, joint types, and joint patterns into a 3D rendering file and a 2D fabrication file (lines 32–36 in Fig. 3). The 3D file can be used to visualize the design. The 2D file can be directly sent to 2D fabrication machines (e.g., laser cutter); thus the resulting fabricated components could be assembled into the desired physical models. Users can repeat steps 1–3 to build designs of arbitrary complexity. Also, hierarchical composition (see Section 4.2) could also be harnessed to create sophisticated furniture models. It is worth noting that we represent all components as 2D polygons since they are defined as planar geometries in our system and only substantialized as 3D structures when fabrication specifications (especially material and its thickness) are resolved.

By leveraging the knowledge of domain experts, our system can release novice from tedious engineering details and underlying system implementation, and instead focusing on functional design of desired

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

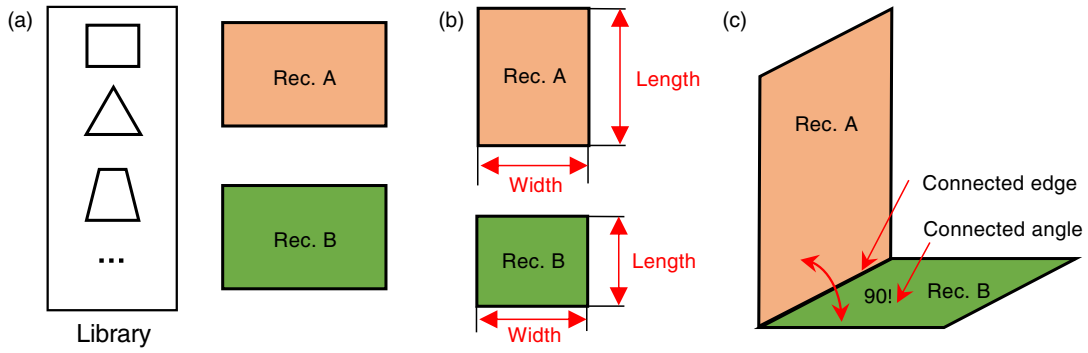

Figure 2. *An illustration of a typical design process in our system. (a) Users select two predefined rectangle components from our library; (b) Users specify the dimension of each component (e.g., widths and lengths of the rectangles); (c) Then they define the connection between the two rectangles at selected edges with an 90° angle.*

```python
# Import the Component and Joint Type
from svggen.api.component import Component
from svggen.api.composables.graph.Joint import FingerJoint

# Initialize the component
Component = Component()

# Select the predefined components
Component.addSubcomponent("A","Rectangle")
Component.addSubcomponent("B","Rectangle")

# Add the dimensional constraints
Component.addParameter("l1", 30, paramtype="length")
Component.addParameter("l2", 60, paramtype="length")

# Specify the dimensional constraint for the components
Component.addConstraint(("A","l"), "l1")
Component.addConstraint(("A","w"), "l2")

Component.addConstraint(("B","l"), "l1")
Component.addConstraint(("B","w"), "l2")

# Define the connection between components
Component.addEdgeToEdge(
    ("A", "t"),("B","b"), orientation="front-front",
    offset = (0,0,0), angle=(90, 0, 0), alignment = "left"
)

# Save the design file for later hierarchical composition design
Component.toYaml("library/ConnectionJointBuilder.yaml")

# Generate the output files for the design
Component.makeOutput(
    "output/ConnectionJointBuilder", tree=False,
    display=False, thickness=3, joint=FingerJoint(thickness=3)
)
```

Figure 3. *The interface of our proposed system, showing how we connect two rectangle components at selected edges with an 90° angle, as presented in Fig. 2.*

furniture by following the above-mentioned steps. For experts, our system has also provided the freedom to (1) create their own new basic components by defining polygons (needs to specify parameters and ports as shown in Fig. 5), (2) to define fabrication specifications, like kerf and amplitude of interference, and (3) to add new joints. It is worth noting that all the three advanced operations are only introduced for completeness of presentation and limited to experts; novices only need to design and build new furniture by composing existing components using the four step process (component selection, parameter constraint, connection definition, and fabrication specification).

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

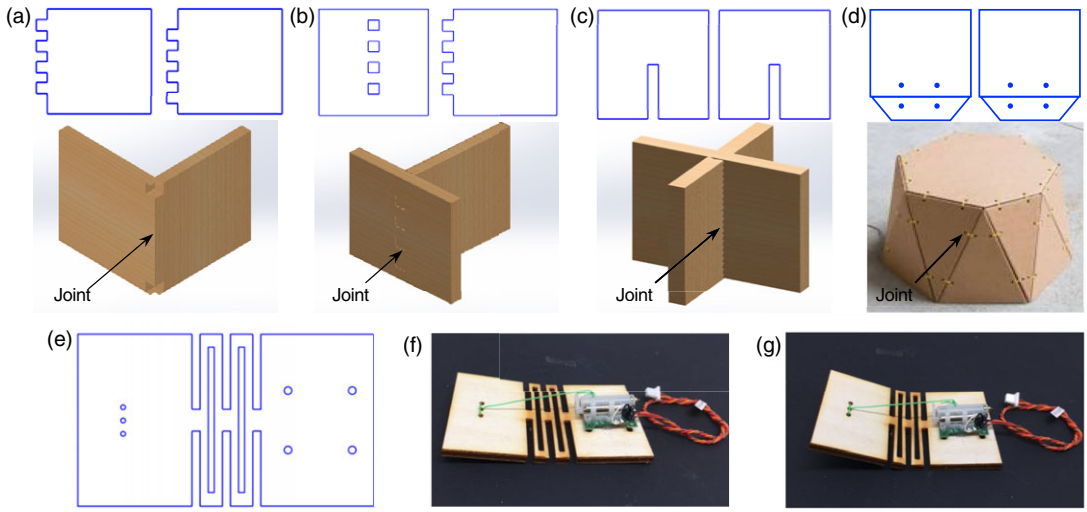

Figure 4. *Joint collection. (a) A finger–finger joint for an edge–edge connection; (b) A finger–hole joint for an edge-face connection; (c) A slot–slot joint for a face–face connection; (d) A flap joint for edge–edge connection [33]. (e) The fabrication pattern of a cable-driven joint. The relaxed and bent states of the joint are shown in (f) and (g), respectively. It is worth noting that cables are needed for both flap joints and cable-driven joints and linear motors are required for cable-driven joints.*

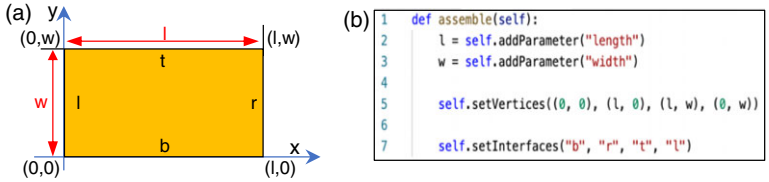

Figure 5. *An example of parameterized abstraction. (a) Rectangle component geometric diagram with parameters labeled; (b) Program implementation of a parameterized rectangle class in Python script in our system.*

3.2. Design space

In our system, connections represent the spatial relationship of the two connected components, while joints are physical implementations of the corresponding connections. Theoretically, our system can design furniture with arbitrary complexity according to users' input due to the availability of our connection architecture. Despite the arbitrary complexity of designs allowed in our system, the design space is confined by the physical limitations of the joints to ensure the manufacturability of the resulting furniture. Currently, we employ three types of joints, that is, finger–finger joint, finger–hole joint, and slot–slot joint (as shown in Fig. 4(a)−(c)), to implement corresponding connections. These joints are instinctively easy to assemble, minimizing the need for careful positioning or hardware-based attachment for casual end-users. These three types of joints are only applicable to 90-degree connections; thus, furniture with non-90-degree joints can be designed in our system but are not valid for assembly. However, this limitation can be removed as long as we can find available joints that can be implemented to fix two connected components at a specific angle, and easy to be assembled by novice users. For example, flap joints can be a good candidate as demonstrated in Fig. 4(d). Likewise, other types of joints can be implemented to expand the design space to enable more functionalities of the resulting models.

Our system is currently specialized for furniture designs with single-type raw materials with uniform thickness. However, our system has the potential to accommodate multiple materials with various

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

thicknesses. For example, we can add two additional parameters, that is, material type (including Young's modulus, Poisson ratio, and so forth) and thickness, into the component class as described in Section 4.1.1. Consequently, our system can modify the length and width of the fingers of joints.

4. System implementation workflow

In this section, we outline the five main steps of system implementation workflow, including parameterized abstraction, hierarchical composition, coordinate placement, intersection autodetection, and output and assembly. (1) All design elements, including components, connections, and resulting models, are parameterizedly abstracted, which allows users to design furniture in a function-based manner; (2) A hierarchical composition algorithm is then employed to enable users to create complex furniture by functionally combining existing furniture models; (3) the 3D coordinates of all components are computed through a coordinate placement algorithm; (4) these coordinates are then fed into an intersection autodetection algorithm to automatically identify necessary connections and thus place joints to corresponding positions according to the input fabrication specifications to finalize the models; (5) the resulting furniture models are output with 2D fabrication files (such as .DXF and .SVG) and a 3D .STL file, ready for fabrication and assembly. In addition, we describe the main operational features of our system in detail.

4.1. Parameterized abstraction

The 3D geometries of flat-pack furniture in our system are defined as a composition of connected components. In traditional furniture design process, creating a piece of functional furniture design can be rather challenging since users may need to adjust many parameters with complex dependencies while maintaining the manufacturability and assemblability. Even with the aid of some CAD tools, users may still experience great difficulties due to the lack of in-depth understanding of CAD software and the manufacturing process.

In order to allow casual end-users to design furniture with ease, our system largely simplifies the design process into an abstracted function-based manner. To achieve this abstracted design, we parameterize all elements in furniture creation process, including components, connections, designated furniture models, and fabrication specifications. Therefore, we introduce the parameterized abstraction of components, connections, and furniture models in following paragraphs.

4.1.1. Components

Representation. As mentioned before, components are represented as 2D polygons. Therefore, in our system, a component's fabrication-related parameters, for example, material type and its thickness, are not defined until specific fabrication and assembly methods are determined by users in the final steps of the design process. At that time, these abstract components are implemented automatically as 3D ingredients according to the input fabrication specifications. More details about how we represent components are described in the next section, where we use a rectangle as an example.

Components in our system fall into two categories: basic components and hierarchically composed components (later introduced in Section 4.2). Our system already comes with a set of predefined, commonly used polygon components, such as rectangle, trapezoid, n-side polygon, and so forth. After abstract parameterization, every component in our system is programed as an object instantiated from a corresponding component class, and represented as follows:

$$\textbf{Component}\ (\textit{para}\ \mathbf{1}, \textit{para}\ \mathbf{2}, \ldots, \textit{para}\ \mathbf{i}, \ldots, \textit{para}\ \mathbf{n})$$

where **Component** is the component class name and *para* **i** is the *i*th predefined parameter of the component. To instantiate a component object of the corresponding component class, users only need to specify the parameters.

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

Construction. Figure 5 uses a rectangle component as an example to demonstrate how we define a component class in our system (each component is an instance of the component class). Firstly, we need to define the parameters of the component class (lines 2–3), which are the length l and width w of the rectangle in this case. Then, we specify the outer vertices of the component class whose order follows the right-hand rule so that the front of the component is facing the out-of-page direction (line 5). Lastly, we set a series of interfaces to the component class, which are some ports that can be used to connect with the interfaces of other components. In this example, we set the edges of the rectangle as interfaces with the name b, r, t, and l. Thus, we have a rectangle class with four interfaces and two parameters. It is worth noting that we have specified the angle between the length l and width w to be 90 degrees due to the definition of a rectangle shape. Otherwise, we could add another parameter, angle a, to specify a parallelogram component, which is a super-set of the rectangle class. In the same manner, we can follow this recipe to build arbitrary 2D polygon classes.

Experts or developers can also define their own custom components following the same workflow illustrated above, which only requires them to specify the parameters, vertices, and interfaces of the components. Customizing components is one way to define desired components when they are not available in our library. This base-level component creation is typically not necessary, though, and presented here for completeness. More commonly and more intuitively, users can harness hierarchical composition (see Section 4.2) to piece together elementary components into new complex component. For example, users can obtain a new "L" shape component by combining two rectangles together along edges. Thus, we theoretically only need a few basic polygons, that is, triangle and rectangle (though it can also be built from combining two right triangles). Therefore, the number of initial designs in the library is not critical; however, it is easier and faster for the nonexpert users to create their design if there are more available basic designs so that they can directly pick-and-place components instead of creating every component from scratch. It is worth noting that experts or developers are allowed to define components with fewer parameters, called meta-parameters, by adding some geometric constraints to the original parameters. Thus, they are not exposed to the enormous design parameters when the design becomes complicated. For example, we can only select the length of the rectangle as the manipulable metaparameters and geometrically constrain its width as a half of its length to create a component with a fixed aspect ratio.

4.1.2. Connection

Representation and construction. Connections in our system are also parameterizedly abstracted. Once a connection has been defined, we build an associated connectivity item to store all its information that is efficient to embody the connection physically in following operations. Later, the abstract connections will be implemented physically to actually join connected components. There are in general two different methods to specify connections between components: (1) users can create connections in a global coordinate frame or (2) in a local coordinate frame. These two methods have their own advantages and disadvantages. In this project, we choose to use the latter so that the connection is defined referring to the local coordinate frame of the existing component instead of the global coordinate frame of the design. In this manner, the connection between the two connected components can be defined easily since the connection itself is the local relationship of the two components. Otherwise, the global design frame would post challenges to the implementation of joints that must accommodate arbitrary connections. This requirement would complicate the structure of designs and may need extra accessories for joints, which would increase the difficulty of fabrication and ① assembly.

A connection is represented by a directed line in a connectivity graph of a furniture model. Here, we use a computer desk, as shown in Fig. 6 as an example. In the figure, connection 1 (② → ①) denotes that component ② is connected to component ① at some interfaces. In the graph, the edges of the line represent the connected interfaces of corresponding components. The direction specifies the way we specify the connection. Note that 1 (① → ②) is different from 1 (② → ①); they may represent the same spatial relationship with specific parameters in the 5-tuple. However, the way we specify a connection does not indicate the assemblability and the order of the assembly. More explanations would be found in the following paragraphs.

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

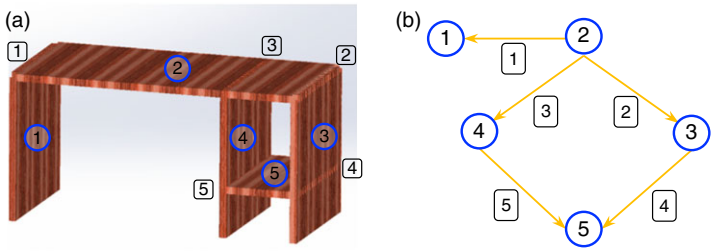

Figure 6. *Representation of a furniture model. We take a computer desk as an example. Components labeled as circled number, for example, ①) and connections as boxed number, for example, 1. (a) 3D model of the desk; (b) Connectivity graph of the desk with connections represented as directed yellow lines, whose directions indicate the connection orientation.)*

Finding a proper way to specify the abstracted connection between two components can be nontrivial because it needs to be accurate, concise, and intuitive for users, while being able to express all possible spatial relations. In our system, the connection is specified abstractly as a 5-tuple. For instance, when component C_A is connected to C_B, we can call the constructor to define the connection:

$$\textbf{Connection}\left(\left(\mathbf{C_A}, I_A^i\right), \left(\mathbf{C_B}, I_B^j\right), P_A, P_O, P_R\right)$$

where C_A and C_B are two connected components. In this expression, the first component C_A is connected to the second component C_B. In this connection, C_A is named as the connecting component and C_B is named as the connected component. Thus, this connection could be shortly annotated as **Connection**($A \rightarrow$ **B**). The sequence of the two components matters. I_A^i and I_B^j represent the selected interface i of component C_A and the interface j of component C_B. These two selected interfaces will be connected together. P_A, P_O, P_R represent the alignment, offset, and rotation of the connection. The value of P_A is either "front-front" or "front-back." "Front-front" means that two connected components have the same orientation while "front-back" indicates their orientation is opposite to each other. P_O is a 3-tuple, specifying the 3D offset vector that component A (connecting component) should travel in accordance with. P_R is also a 3-tuple, defining how component A along with its local coordinate system is rotated around its own x, y, and z axis, respectively. For example, in Fig. 6(b), the connection between component ① and ② can also be described as **Connection**(② → ①).

Using this 5-argument connection constructor, we can represent any possible spatial relations between two components while providing users an intuitive and easy way to state connections. Here, we will illustrate this connection constructor in detail (see Fig. 7). For instance, **Connection**($(A, t), (B, b), front - front, (v_x, v_y, v_z), (\theta_x, \theta_y, \theta_z)$) can be visualized as in Fig. 7(b). From the connection constructor, we know that interface t of component A is connected to interface b of component B. Since the alignment is specified as "front-front," the orientation of both faces facing up results in a temporary position as the left graph in Fig. 7(b). x-y-z is the coordinate of component B being set as the global coordinate for this connection while x'-y'-z' is the local coordinate of component A, fully overlapping with B's. Then a 3D offset vector (v_x, v_y, v_z) is applied to component A (as the middle graph shows) and followed by a 3-Axis rotation $(\theta_x, \theta_y, \theta_z)$) to change the orientation of the component (e.g., (90,0,0) represents a 90° rotation about x axis as in the right graph in Fig. 7(b)). The other case with "front-back" alignment is also presented in Fig. 7(c). Though rather simple and intuitive, the connection specification process could be greatly simplified with a graphic user interface later on.

With these well-defined connections between components, each furniture design can be organized into a connectivity graph (Fig. 6(b)). Thus, the position of each component can be computed by tracing this connectivity graph. For example, we can compute the position of component ② with the **Connection**(② → ①) and the position of component ① (which is known). In the same manner, we can calculate the positions of the rest of the components accordingly. It is worth noting that we can have two

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

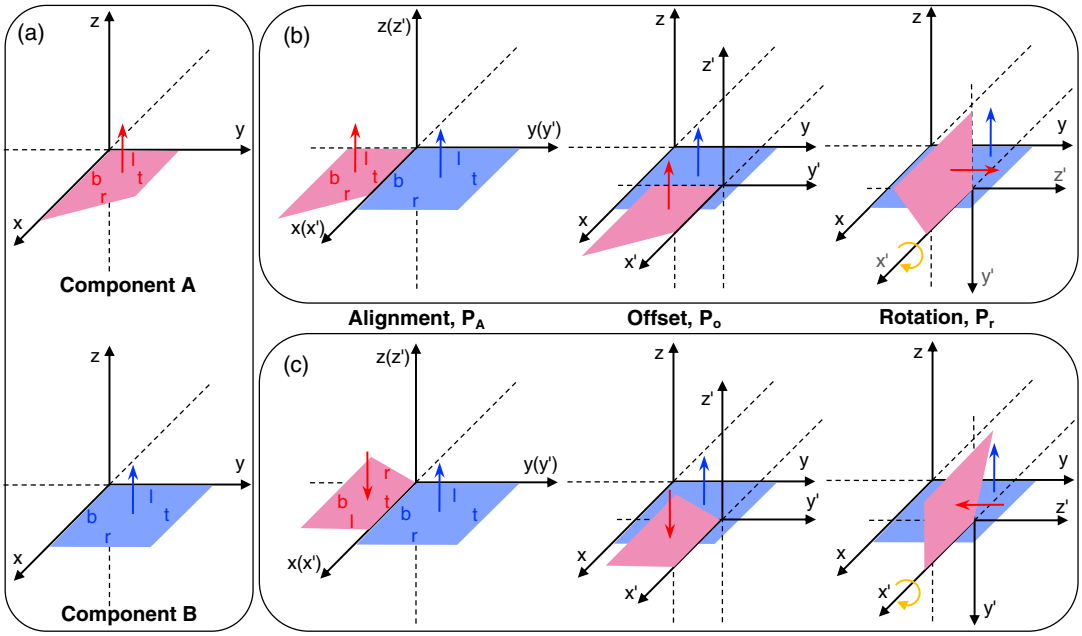

Figure 7. *Connection visualization. (a) Component A and B with their interfaces labeled and original coordinates specified; Connections with "front-front" alignment (b) and "front-back" alignment (c). Left: Alignment defined; Middle: 3D offset; Right: 3-axis rotation.*

different expressions to represent the same spatial connection. In other words, these two different expressions are interchangeable and depict the same spatial relationship. For example, **Connection**(②→①) is different from **Connection**(①→②). However, they characterize the same 3D relationship that two components are connected at the edge orthogonally (see Fig. 6).

Physical implementation. Though parameterized abstraction can greatly facilitate the design process, the connections need to be implemented physically to actually join connected components when it comes to manufacture. In this paper, we decide to use planar joints to embody the connections to reduce the difficulty of fabrication and assembly. Thus, users do not need to go through the tedious assembly process with system-defined joints [19].

To enable the manufacturability of furniture designs, joints must be added at places of intersections. However, for casual end-users, determining and drawing proper joint patterns can be a liability, even with the help of some design software (e.g., AutoCAD, Solidworks, UG, Inventor, Inkscape, and so forth). Our computational design tool will automatically add one of the three types of joints (as shown in Fig. 4(a)−(c), respectively) according to specific abstract connections. More details for these three types of joints can be found in Appendix A.1). The joints that we introduce here have the following two advantages: (1) they can be easily fabricated using modern 2D manufacturing tools (e.g., laser cutter, waterjet, and jigsaw); (2) are handy to assemble even without skilled craftsmanship.

Theoretically, any joints, especially those that satisfy the above requirements, can be incorporated into our system since the joints are merely the physical implementation of abstracted connections [34]. For example, the flap joints [33] (see Fig. 4(d), not implemented in our system) could potentially be adopted to achieve edge–edge connections. Thus, the system would generate holes for connections instead of finger patterns to implement joints. However, this type of joint requires more labor and skill to assemble. Similarly, fasteners (e.g., nails) or adhesives (e.g., glue) can be used in additional types of joints; this would increase the difficulty of assembly as well [29]. It is straightforward to extend our library of joints to expand the design space, though care must be taken to balance against the capabilities of a proposed user.

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

In addition, active joints could be possible to realize connection implementation, which leads to active devices, such as active furniture and robots. As a proof of concept, we create cable-driven joints to allow resulting devices to have angular movements. For example, we connect two rectangle components with a cable-driven joint, as shown in Fig. 4(f) and (g). The fabrication pattern is shown in Fig. 4(e). By adding a lattice pattern [35] (more details can be found in Section A.2) along the connected edges of the two rectangles, a flexible joint is formed to allow angular movements between two connected components. A linear motor with its shaft is tied to the left rectangle using a cable, is then attached to the right rectangle, as shown in Fig. 4(f). The back-and-forth movement of the shaft of the motor will change the angle between two components (see Fig. 4(f) and (g)). Presumably, this type of joint can be used as living hinges for doors of furniture or movable joints of robots.

In this paper, joints are mainly assembled through interference fit—also known as a press fit or friction fit—a form of fastening between two tight-fitting mating parts that produces a joint held together by friction from a compressive normal force. As the amplitude of interference (amount of geometric overlap between mating parts) grows, both the deformation of the joint material and the difficulty of assembly increase, which means optimal interference amplitudes are needed to be calculated based on the fabrication specifications. Our system can automatically output the optimal profiles of joints once the fabrication specifications (e.g., materials and fabrication machines) are defined by users. More details could be found in Appendix A.1.

4.1.3. Model

Using the connection constructor specified on relevant components, an abstract design model is built internally to store all the information (including components with specific geometry constraints and connections between components) relevant to the design. This designed model can be visualized as a directed connectivity graph. Here, we use a computer desk as an example. As shown in Fig. 6(a), the computer desk consists of five rectangle components and five connections. The connectivity graph of the computer desk is shown in Fig. 6(b). Each component is labeled as a circled number, such as ① and each connection is represented by a numbered symbol, for example, 🄀. Each directed edge denotes a connection with its vertices intersected with circles at their corresponding interfaces. For example, component ① is connected to one interface of component ② resulting in connection 🄀 while component ③ is connected to another interface of component ② to form connection 🄀. Thus, a connectivity graph effectively includes all information relevant to the corresponding design. By traversing the graph following the algorithm presented in Section 4.3, we can generate the 3D coordinates of each component of the design for further operations. Each connectivity graph, representing a design, can also be stored in our library as a .YAML file, which can be in turn used as a new component to hierarchically compose more complex designs in a function-based manner.

4.2. Hierarchical composition

4.2.1. Design principle

To enable users to build complex furniture designs with ease, we harness function-based hierarchical composition, which allows users to recursively build up to the desired complex furniture constructions from relatively simple existing designs. This means we organize all the parameterized data of the furniture design into a hierarchical tree, whose structure is defined by how we recursively create the furniture. In this manner, functional models at any level could be treated as components to specify higher level furniture designs. For instance, to build a bunk bed as illustrated in Fig. 8(d), instead of building the whole piece from scratch, users can simply select these existing designs, a computer desk, a bed, and a ladder (as shown in Fig. 8(a)−(c), respectively) to be composed together. The computer desk, bed, and ladder can be further decomposed into elementary components (e.g., rectangle). Therefore, the i_{th} level hierarchy M^i of a hierarchically composed model can be written as:

$$M^i = \left(\left\{Component^i\right\}, \left\{Connection^i\right\}, \left\{Constraint^i\right\}, \left\{Interface^i\right\}\right)$$

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

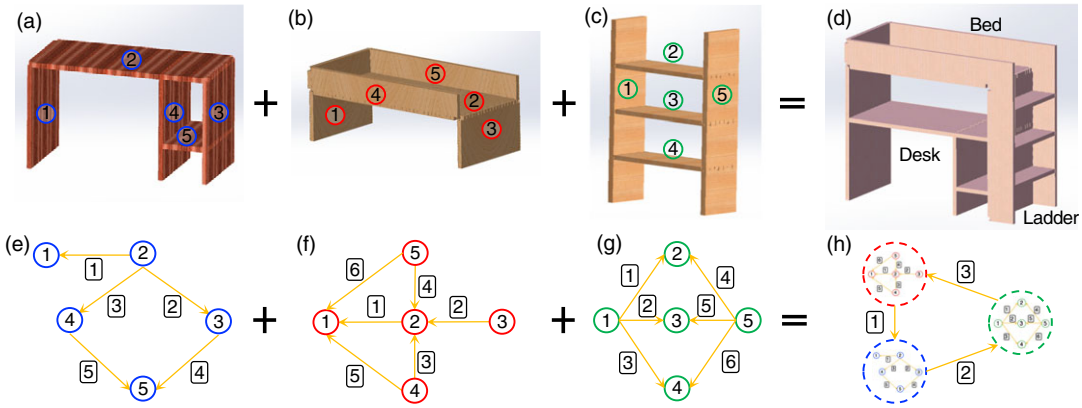

Figure 8. *A illustration of hierarchical composition with a bunk bed. Three "components," that is, a computer desk (a), a bed (b), and a ladder (c) are composed into a bunk bed (d). Each "component" itself is a furniture design with certain functionalities. The connectivity graphs of three "components" are also pictured in (e), (f), and (g). The final connectivity graph of the bunk bed is also directly composed of all "components" H).*

where $\{Component^i\}$ is a set of "basic" components consisting of the hierarchical design in this level. Each "basic" components in this set can be a composed furniture design or an elementary component. $\{Connection^i\}$ is a set of connections specifying how the "basic" components are connected, $\{Constraint^i\}$ is the set of parameters constraining model M^i, and $\{Interface^i\}$ is the set of interfaces of the new hierarchically composed model M^i.

To further demonstrate how we implement our system, we use the above mentioned bookend (see Figs. 2 and 3) as an example. Each rectangle (A and B) is an instance of the Rectangle class, which is defined in Python. After parameter constraint and connection definition, we can obtain a composite structure—two rectangles connected perpendicularly along an edge. Due to the hierarchical architecture of our system, this composite structure can itself be used as a component, ConnectionJointBuilder, by saving it as a .YAML file into the library (see lines 29−30 in Fig. 3). This new class of component, ConnectionJointBuilder, inherits the properties from the composite structure with two rectangles connected perpendicularly along an edge. Recursively, we can use this hierarchical composition strategy to build furniture with arbitrary complexity with ease, as shown in Fig. 8.

4.2.2. Model reconstruction

Each design in our system is represented and stored as an abstract connectivity graph. When users compose several simpler designs into a more complex functional model, all connectivity graphs will be added to a new high-level connectivity graph as per the connections specified at this hierarchy by the user. This means that all information of each simpler design is integrated to generate a new hierarchical data tree with its own components and connections preserved, as shown in Fig. 8(e)–(h). In the same manner, this hierarchical data tree will be integrated as a branch of a new tree at a higher hierarchy when current design is composed into a more complicated furniture.

4.3. Coordinate placement

To place each component in a global coordinate system and find the places of intersections to add proper joints, we employ a coordinate placement algorithm. At the design stage, we require users to define connections between components. However, these connections only specify the relative spatial relation, which is not necessarily the positions at which joints are placed. Therefore, we use an intersection

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

Algorithm 1. Compute the global 3D coordinates for every component of the component set {C} of a design based on the defined associated connections

1: Randomly select a component C_i from {C}
   //Return coordinates of components connected to C_i
2: **function** FIND3D(C_i, T_{global})
3: $C_i.T_{3D} \leftarrow T_{global}$
4: **for Connection**($A \rightarrow$ **B**) in $C_i.Connections$ **do**
   //C_i is connected component
5: **if** C_i *is* C_B **then**
6: $T_{relative} \leftarrow FindRT(C_{A \leftarrow B})$ ▷ Find relative transformation matrix
7: $T'_{global} \leftarrow T_{global} \cdot T_{relative}$ ▷ Calculate new global transformation matrix
8: return Find3D(C_A, T'_{global})
   //C_i is connecting component
9: **else if** C_i *is* C_A **then**
10: $T_{relative} \leftarrow FindRT(C_{A \leftarrow B})$ ▷ Find relative transformation matrix
11: $T'_{global} \leftarrow T_{global} \cdot T^{-1}_{relative}$ ▷ Calculate new global transformation matrix
12: return Find3D(C_B, T'_{global})

autodetection algorithm (see Section 4.4) to place necessary joints based on the derived spatial relation to finalize the design for manufacturing and assembly. Having derived the connectivity graph representing the furniture designs, our system can perform a traversal on the graph to recursively compute the 3D coordinates of each component following Algorithm 1, which will later be used to build up the 3D model.

Initially, each component is placed within its own local coordinates as defined by the user as in Fig. 5(a). Our goal is to find each component's 4 by 4 transformation matrix, T_{global}, which transforms the homogeneous coordinates of the original components into a global 3D coordinate space as per the connections. To do so, we adopt the recursive Algorithm 1. The input of the function consists of the component set C, a 4 by 4 transformation matrix T_{global} that represents its global coordinate. The algorithm randomly selects a component C_i as a starting point and recursively find the coordinate of all components connected to it. This step starts by storing the transformation matrix as an attribute of the component. Then, for each connection, for example, **Connection**($A \rightarrow B$), that involves this component, if this component C_i is connected to component C^B (i.e., $C_i = C_B$, line 5), we can find the relative transformation, $T_{relative}$, through the function, *FindRT*(**Connection**($A \rightarrow B$)), between them and times the global transformation matrix of C_B ($= C_i$) to obtain the new global transformation matrix T'_{global} to run the next iteration until the 3D coordinates of all components are calculated (lines 6–8). Another execution will be made for C_A. If C_i is connecting component C^A (i.e., $C_i = C_A$, line 9), then we should find the inverse of the relative transformation between them and times the global transformation matrix of C_A ($= C_i$) to have new global transformation matrix T'_{global} to run the next iteration (lines 10−12).

4.4. Intersection autodetection

After the coordinate placement phase illustrated in Section 4.3, we derive the global 3D coordinates for all components, but the design is still by no means manufacturable. In other words, we need to detect the places of intersections and add proper joint mechanisms to finalize the model as well as to ensure its manufacturability. By performing an automatic intersection detection, we release users from the burden of specifying all places of intersections with connections explicitly. More specifically, using

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

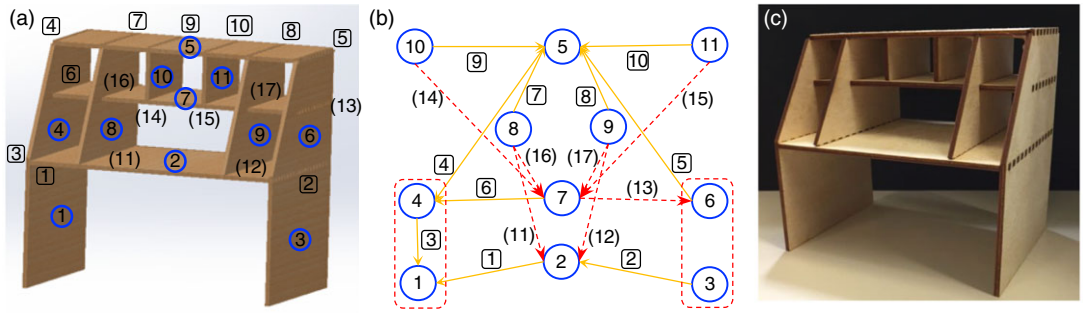

Figure 9. *An illustration of intersection auto-detection. Components labeled as circled number, for example, ① and connections as boxed number, for example, 1. (a) 3D model of a reading desk; (b) Connectivity graph; (c) Fabricated and assembled (scaled) reading desk with 3 mm plywood. Note: This desk can also be built through hierarchical composition, which will be discussed later in Section 5.2.*

the connection constructor, user can easily build the 3D model for their designs with a minimal number of connections, which may not include all necessary intersections. Then our algorithm can automatically handle the intersection detection to guarantee the manufacturability and functionality. This algorithm is particularly useful for hierarchical composition because it is very challenging for casual end-users to specify all necessary connections thoroughly due to the geometry complexity. Hence, it help users to focus on design in a function-based manner.

In this section, we use a reading desk, as shown in Fig. 9(a), as an example to illustrate the algorithm. To design such a reading desk with 11 components and 17 joints, users only need to specify 10 connections (from 1 to 10 in Fig. 9(b)) with other necessary intersections (or joints) (from (11) to (17)) inserted automatically by our algorithm. For instance, a joint is automatically added between component ② and ⑧ while there is no connection defined by users. Moreover, our algorithm can help to merge coplanar components to reduce the complexity of design and assembly. For example, component ① and ④, originally connected by connection 3, are merged into a single component.

In the following two sections, we will explain the intersection autodetection algorithm in detail with which divided into two parts: coplanar faces merging (lines 1–6), intersection segments searching (lines 7–16), and joints inserting (lines 17–27).

4.4.1. Coplanar faces merging

Frequently, several different components of a furniture design end up being coplanar (or overlapping). Merging them often not only simplifies the design but also lowers the difficulty of fabrication and assembly. For coplanar faces merging, we iteratively select two components C_i, C_j from the components set $\{C\}$ (line 4) and determine if they are coplanar faces with intersections, which is fairly easy as we have already found the 3D coordinates of all components. Then, we use a geometry boolean operation library to find the union of the 3D global coordinates of C_i, C_j (line 5). After finding the union of 3D global coordinates, we need to transform them into the local 2D coordinate system of the planar component so that we can later work on the 2D output file for the design. Therefore, we will dot product the union of the 3D global coordinates with the inverse of the 3D transformation matrix of one component C_i (line 6) and set the vertices of the other component C_j (line 7) to be empty. This means that all information of component C_j is transferred into component C_i.

4.4.2. Intersection segments searching

For any two components C_i, C_j that are not coplanar or parallel, we will first find the intersection line l of the two 3D planes that these components are lying in (line 11). Then, we find the two sets of segments where the intersection line intersects with component C_i, C_j (lines 12−13). After finding the union of

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

Algorithm 2. Find all necessary intersections and joints of a designed model composed of a set of components {C} and specified connections

1: **function** IntersectionDetection($\{C\}$)
    //Merge coplanar faces
2: **for** C_i in $\{C\}$ **do**
3: **for** C_j in $\{C\}$ **do**
4: **if** $coplanar(C_i, C_j) \wedge C_i \cap C_j \neq \emptyset$ **then**
5: $V_{i\cup j} \leftarrow (C_i \cup C_j)$ ▷ find the geometric union of C_i, C_j
6: $C_i.V^{2D} \leftarrow C_i.T^{3D^{-1}} \cdot V_{i\cup j}$ ▷ convert to global coordinate system
7: $C_j.V^{2D} \leftarrow \emptyset$
    //Find segments of intersections of each component
8: **for** C_i in $\{C\}$ **do**
9: **for** C_j in $\{C\}$ **do**
10: **if** $\neg coplanar(C_i, C_j) \wedge \neg parallel(C_i, C_j)$ **then**
11: $l \leftarrow C_i \cap C_j$ ▷ find the line of intersection between C_i and C_j
12: $\{s_i\} \leftarrow l \cap C_i$ ▷ find segments of intersection between l and C_i
13: $\{s_j\} \leftarrow l \cap C_j$ ▷ find segments of intersection between l and C_j
14: **for** s_k in $\{s_i\} \cap \{s_j\}$ **do**
15: $s_k^i \leftarrow C_i.T^{3D^{-1}} \cdot s_k$ ▷ convert to global coordinate system
16: $s_k^j \leftarrow C_j.T^{3D^{-1}} \cdot s_k$ ▷ convert to global coordinate system
                //Add joints according to the types of intersections
17: **if** $l_i \in edge(C_i) \wedge l_i \in edge(C_j)$ **then**
18: $C_i.finger \leftarrow C_i.finger \cup \{l_i^1\}$
19: $C_j.finger \leftarrow C_j.finger \cup \{l_i^2\}$
20: **else if** $l_i \in edge(C_i)$ **then**
21: $C_i.finger \leftarrow C_i.finger \cup \{l_i^1\}$
22: $C_j.hole \leftarrow C_j.hole \cup \{l_i^2\}$
23: **else if** $S_i \in edge(C_j)$ **then**
24: $C_i.hole \leftarrow C_i.hole \cup \{l_i^1\}$
25: $C_j.finger \leftarrow C_j.finger \cup \{l_i^2\}$
26: **else**
27: $C_i.slot \leftarrow C_i.slot \cup \{l_i^1\}$
28: $C_j.slot \leftarrow C_j.slot \cup \{l_i^2\}$

these two line segment sets (line 14), we will also perform coordinate system transform step, similar to line 6, to transform the coordinates from the 3D global coordinate system to the local 2D coordinate system of C_i and C_j (lines 15–16). So far, all of the intersection segments within a certain design are identified, ready for joints to be inserted in next step.

4.4.3. Joints inserting

The last step is to determine the types of joints for each component based on the positions of intersection segments. If the intersection segment of a connection is on the edge of both connected components, then a finger–finger joint will be added on them (lines 17–19). If the intersection segment is on the edge for one component and within the face of the other, a finger–hole joint will be selected. Fingers will be added on the former component, and holes will be added on the latter (lines 20–25). If the intersection segment is within the face of both components, then a slot–slot joint (lines 26–28) will be added on them. All

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

inserted patterns of joints are automatically calculated by considering fabrication specifications (more details about the pattern generating can be found in Appendix A.1).

4.5. Output and fabrication

Each furniture design in this system is an instance of a common parent class, stored as an executable Python script (see Fig. 3). At execution time, running this script places all the components into a final 2D design file (.SVG or .DXF)—as shown in Figs. 4 and 15—to be sent directly to the 2D cutting machine (e.g., laser cutter or waterjet) or printed onto a pattern for hand cutting. During this process, the fabrication specifications are considered, and joints are selected and rendered in the final files. Components are placed separately from each other to avoid overlapping. However, we can manually rearrange components (or even divide components into two separate files) if the cutting space cannot accommodate all components. In addition, we need to manually split (at a position away from the original edge for connection) and fabricate long pieces that do not fit within the cutting space; then combine split pieces back together by using coplanar finger–finger joints.

To enable easy and fast fabrication, we adopt planar joints that can be cut through the 2D cutting process. A typical manufacturing specification is a material with its corresponding thickness. For finer grained expert control, additional fabrication specifications could adjust laser cutting kerf and desired magnitude of friction-fit interference. A specific joint pattern is chosen appropriate to these specifications (see Section A.1). To ensure successful manufacturability, our system automatically evaluates some geometrical considerations about the connected components and corresponding joints: (1) the dimension perpendicular to the connected edge of components must accommodate the finger length and (2) the dimension along the connected edge of components must accommodate sufficient finger widths. More details are given in Section A.1; our system prompts warnings when these conditions are not satisfied.

In this way, our system as mentioned above assigns each component a unique manufacturing specification that dictates how the design specifications translate to real-world realizable output files. These 2D files are also available for paper cutters (e.g., Silhouette CAMEO 2) and CNC router (via conversion of the .DXF files to .IGES or .G-code). Our system also can render 3D models (.STL files, see Figs. 6(a), 8, and 9(a) for visualization to assist design process. If necessary, these .STL files can be used as inputs of a 3D printer to directly generate 3D mockups. This complete process is validated through several successful designs (see Figs. 11, 13, and 14).

4.6. Validation and assembly

Though some other design tools simplify the design process, the assembly processes still require a lot of experience, which inhibits novice users from creating their own custom furniture (see Table I). Our solution for this issue is to adopt planar joints that are realized exclusively through the single 2D cutting process and the easily assemblable planar joints as mentioned in Section A.1. These planar joints are easy to align due to the particular finger configuration and are easy to assemble through interference fit without the requirement for extra carpentry skills, tools, or materials.

To validate the assemblability and generate an assembly plan of the resulting furniture, we currently provide users a testing process with low-cost substitutes (e.g., copy paper or acrylic sheet) to build small-scale prototypes; this yields intuitive tactile feedback for novice users, without requiring them to map digital/computational representations of the physical assembly steps. Here, we use two furniture designs that have been shown before in the paper as examples. For instance, the original design of the reading desk is not assemblable due to the interlocking between components ⑤ (or ⑥) and ⑨ as shown in Fig. 10(a) and (b). By modifying the geometry of components ⑧ and ⑨ into convex shape, we obtain an assemblable reading desk as shown in Fig. 10(c) and (d). Also, by experimenting with the prototype, we can explore and find some valid assembly plans. One of the valid assembly plans is to (1) install components ⑤–⑧ onto ⑨; (2) add ② and ④ to the assembled structure; (3) finalize the assembly by fixing ① and ③ onto the designed position. Also, it is possible to over constrain the design due to

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

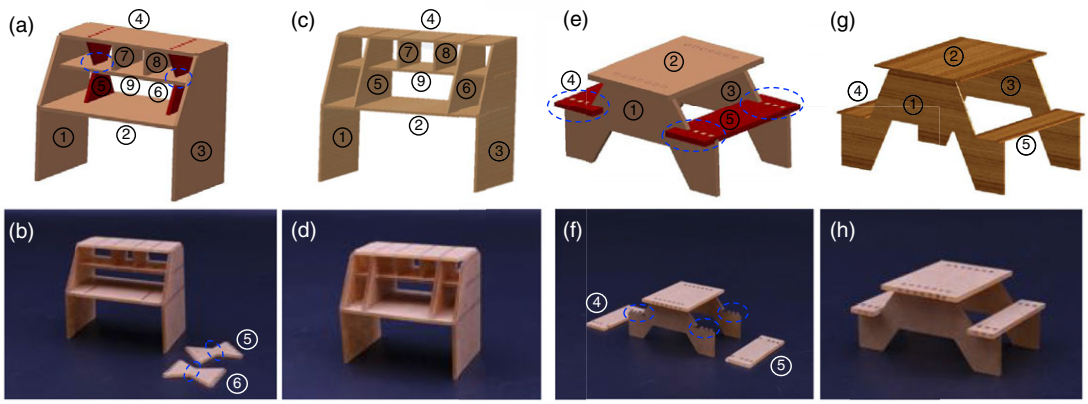

Figure 10. *Examples of assemblability testing by using acrylic sheets (thickness, 1.5 mm) as the low-cost substitutions of lumber. The components that are nonassemblable are in red color.*

redundant connections. As shown in Fig. 10(e) and (f), the two connections between components ① (or ③) and ⑤ (or ④) of the picnic table are incompatible, which makes the design nonassemblable. By removing one of them, the table can be easily installed (see Fig. 10(g) and (h)). Similarly, this validating method can be used to explore the assemblability and assembly plan for other furniture designs due to the low-cost and rapid prototyping provided by our system.

5. Results and discussion

To demonstrate the proposed system in this paper, a variety of (scaled) furniture are designed, manufactured, and assembled. In this section, in order to have a quick demonstration, we choose 3 and 6 mm plywood as raw materials to fabricate (scaled) furniture. The fabrication of real size furniture is very similar. We use a laser cutter to pattern these materials and assemble components manually through interference fit (see Section A.1). Full-sized furniture could be created by using a waterjet on thicker plywood; a similar assembly process could be used to assemble those pieces.

5.1. Design examples

We start with a collection of rather simple furniture designs (see Fig. 11) by simply connecting existing elementary components together. These components are typically simple geometric faces, such as triangles, squares, and polygons. In most cases, users only need to define a small number of connections, which can be far less than the number of joints. For example, users only need to specify six connections to define the spatial relations for all components of a stool in Fig. 11(h). Our system will automatically place all 42 joints (slot–slot joints), which greatly lowers the difficulty of creating this furniture design. The same phenomenon can be found in most of the designs as shown in Fig. 11.

In addition, our intersection detection algorithm can help to merge coplanar components to reduce the number of components of a specific design, which lowers the difficulty of fabrication and assembly for casual end-users. For instance, in Fig. 11(e), two trapezoids are combined into a L -shape component due to coplanar faces merge, which means we have fewer components and joints. Also, the usage of built-in planar joints significantly reduces the number of specifying elements (in particular joints) for similar furniture designs [19] (also see Figs. 13 and 14). This reduction can make the design, fabrication, and assembly processes much easier. Lastly, every model in our system has a large number of continuous design parameters, which allows users to freely customize their furniture to match their desires. After specific constraints added to the original design parameter set, a handful of meta-parameters are obtained to allow for design manipulation with structure preserved. For instance, the simple table in Fig. 11(g), has 4 structure-preserved design parameters, including the length, width, height of the table, and the width

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

Figure 11. *Furniture design examples using our system. The numbers of components, available design parameters, connections, and joints of each design are labeled. For design parameters, the structure-p-reserved values are calculated after some constraints are added to preserve the structure of the designs while the maximal values are included in brackets.*

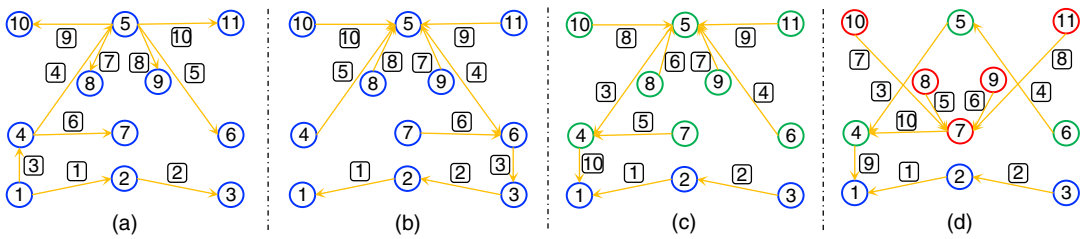

Figure 12. *An illustration of the flexibility of modeling process in our system. Components labeled as circled number, for example, ⓘ) and connections represented by boxed number,for example, □. Components from different furniture models are differentiated by color. (a) Opposite connection direction but the same ordering; (b) Different connection ordering; (c) Hierarchically composed from two different designs; (d) Hierarchically composed from three models.*

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

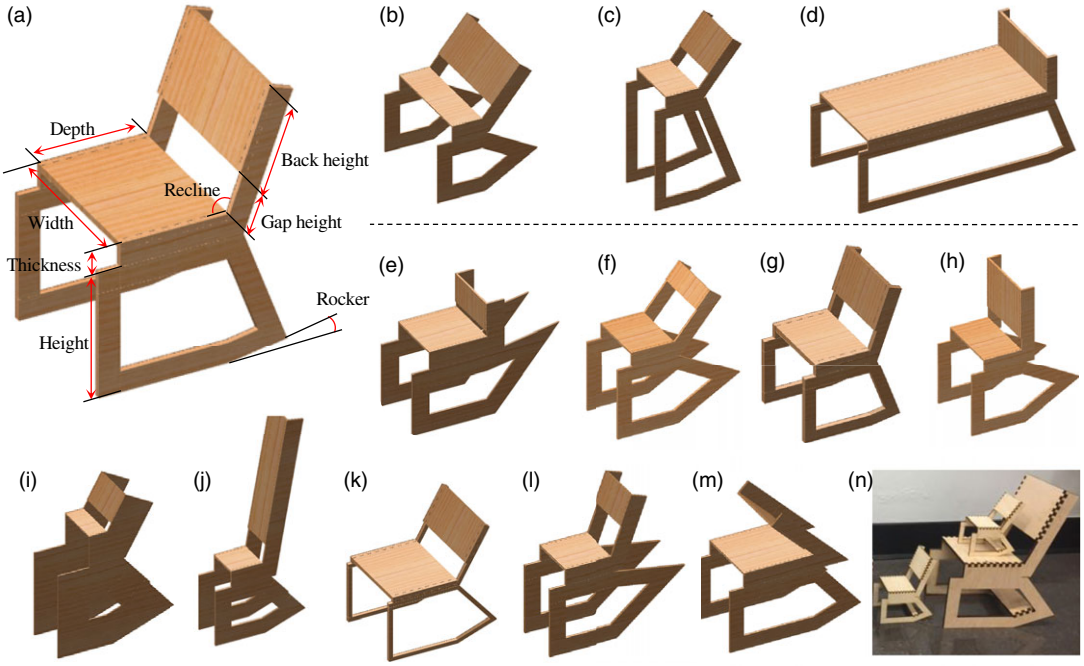

Figure 13. *An illustration of design manipulation after a rocker chair has been composed in our system. Eight metaparameters of a rocker chair are labeled (a). A bunch of variants of the rocker chair are created by manipulating the metaparameters. These variants included a long bench for the side of the pool (b), a tall chair for bar (c), and a rocking bed (d). Other wildly modified rocker chairs are shown in (d)–(m) to demonstrate the vast design space. Several scaled chairs are fabricated and assembled (n).*

of table leg. However, an unconstrained table can have up to 22 parameters allowed to be modified to generate a much broader set of designs.

5.2. Flexibility of modeling process

It is important that the modeling or designing process of a furniture is flexible and robust, which allows casual end-users to freely design a furniture design as they prefer with any ordering or approach. Enabled by the abstract scheme and unique intersection algorithm (see Section 4.4), users can obtain the same desired model (along with fabrication file) with different design processes (e.g., different order or hierarchy). We use the reading desk (see Fig. 6(a)) as an example to show how users can build a furniture differently. Users can follow the same connection order as the original design but flip the direction of every connection (see Fig. 12(a)). For instance, connection 1 is flipped from 1 (② → ①) to 1 (① → ②)). Still, users obtain the same final model as previous. Users can use a more intuitive order of connections as they prefer. One of the example orders is shown in Fig. 12(b). In addition, as illustrated in Fig. 12(c), users are enabled to hierarchically compose the reading desk from two simple furniture models, such as a simple table (composed of components 1–3) and a trapezoid shelf (composed of components 4–11). Similarly, the desk can be composed from three low-level models (see Fig. 12(d)). This high flexibility of modeling process allows users to only focus on design itself instead of tedious engineering details.

5.3. Design manipulation

Owing to the parameterized modeling scheme, users are allowed to manipulate parameters of a design, which means users are capable of modifying a furniture's dimensions while preserving its overall

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

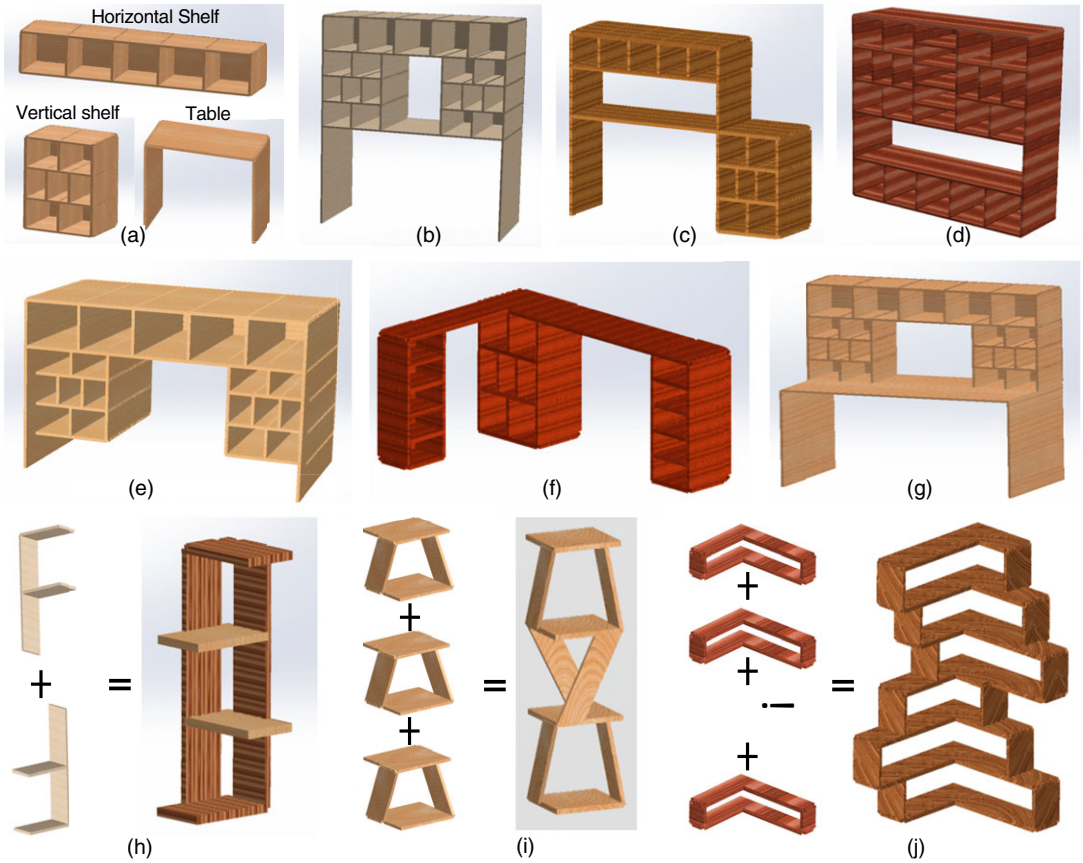

Figure 14. *Compound furniture designs hierarchically composed from existing furniture models. (a) A collection of simple furniture models, including a vertical shelf, a horizontal shelf, and a simple table. Six compound furniture models, composed from the aforementioned simple designs are displayed from (b)−(g). They are a TV console (b), a study desk (c), a multiuse shelf (d), an over-the-toilet storage (e), a corner workshop bench (f), and a dresser (g), respectively. Several other compound designs are also presented in (h), (i), and (j).*

structure. Design manipulation is permitted at all levels of hierarchy. Therefore, the user can make higher-level modifications by editing the hierarchical composition and make more detailed changes by selecting low-level internal nodes. Figure 13 shows an example of how users can continue to explore the design space of a furniture model. For this rocker chair design, eight metaparameters are defined by adding some geometric constraints to greatly reduce the design parameters for structure-preserved manipulation. By varying these eight parameters, users can have a bunch of variants that have distinct functionalities for various applications. For example, users can widen the chair to obtain a long bench for the side of a pool, as shown in Fig. 13(b). Users are also allowed to modify the parameters wildly to create extreme designs, as shown in Fig. 13(e)–(m), for special applications. In addition, some scaled rocker chairs are fabricated and assembled to validate the design manipulation (see Fig. 13(n)).

For users with more expertise, they can impose constraints between related parameters of its constituent parts. For example, we can add constraints between two edges to make them always equal to each other to simplify the design process if they are supposed to be equal. Also, casual end-users can benefit from the constraints encoded by experts by narrowing down the parameters of designs, as the above demonstrated metaparameter definition.

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

5.4. Compound designs

Though the users can always design their furniture from scratch, it is extremely challenging to do so for those who do not possess domain skills. Our hierarchical scheme will save them by composing complex furniture from simpler ones. This hierarchical implementation resolves the challenge by breaking down the complicated design process into recursively combining the relevant simpler building blocks. Figure 14 demonstrates how users can compose several simple designs into more complicated compound furniture models. The building blocks are vertical shelves, horizontal shelves, and simple tables (see Fig. 14(a)). The constructions of these building blocks are rather easy. By hierarchically combining these building blocks, we can obtain numerous various furniture models as shown in Fig. 14(b)–(g). Basically, users merely need to specify how these building blocks are placed against each other, leaving the tedious engineering implementations, such as joints intersection, to our system. Take a study desk as an example (see 14C). Firstly, users stack two tables vertically. Secondly, users add a horizontal shelf on the top of the previously composed design. Lastly, users attach a vertical shelf on the right sideboard of the bottom table to finalize the study desk design. Other compound designs can be created in a similar manner, which greatly releases casual end-users from the tedious work. As you can see, this hierarchical process decomposes the complex modeling into a series of trivial composition steps. The power of this hierarchical approach is also validated by other several hierarchically composed furniture designs shown in Fig. 14(h), (i), and (j). Combining several basic units can result in sophisticated designs. For example, as shown in Fig. 14(i), a fancy bookshelf can be obtained by stacking three identical stools.

The design and assembly of the above-mentioned complicated furniture models were conducted by nonengineering developers of our system. These developers were not familiar with mechanical tolerances, dimensioning, joint configuration, fabrication tools, kerf, and so forth. They were even not familiar with Solidworks or other CAD environments and have very limited hands-on skills, for example, machining. However, they were able to design and create furniture models with the help of our system, which illustrates the feasibility of our system for novice users. In addition, we have involved several nonengineering volunteers during the development phase and improved our system regarding the collected feedback.

6. Conclusion and Future Work

In this paper, we have proposed a computational, function-based design pipeline for digital fabrication of flat-pack furniture. Our system enables casual end-users to easily design, customize, fabricate and assemble furniture models by leveraging parameterized abstraction, hierarchical composition, intersection autodetection, and planar joint design. Specifically, the finger-based design of the joints allows an easier alignment between the components; it also obviates the need for adhesives or fasteners to join two pieces together. For example, carpentry skills (including but not limited to alignment, position holding, and nailing) and tools (such as nails, screws, and markers) are usually necessary for connecting two pieces together (see Table I). However, in our system, the user only needs a mallet (for press fit connection) and minimal carpentry skills, which can greatly reduce the difficulty and enhance the accessibility of furniture creation. Moreover, thanks to the template-free design scheme, no predefined models are needed, which further reduces the dependence of design on domain experts [19]. We have demonstrated the power of our approach by designing, fabricating, and assembling various (scaled) furniture models. Our method also shows the potential of introducing the design of customizable furniture to the general public by greatly reducing the required resources. In conclusion, we present a fully computational design tool for casual end-users to design their own flat-pack furniture that are guaranteed to be manufacturable and easy to assemble. We believe that our work, together with the flat-pack furniture library we release, will inspire interesting future studies. For instance, building a variety of graphical and visual interfaces atop the scripted design language could explore interactions to further facilitate the accessibility of the design process.

A natural extension of our pipeline is to incorporate physical simulation. There are many mechanical properties of furniture designs that need to be investigated in order to guarantee their functionality.

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

These properties include strength of joints, stability, and stress distributions of furniture models under typical loading in daily life. Given the abstract and hierarchical scheme of our system, components and designs are frequently reused to build more complex models, which suggests a data-driven approach to accelerate the physical simulations.

Currently, the mechanical structural performance of the designed furniture can only be validated after fabrication and assembly. Therefore, to guarantee the strength of the joints and minimize risk of failure in the assembled furniture, we intentionally enlarge the amplitude of the interference between fingers of joints. This heuristic notably improves the performance of designed furniture in practice. However, this method increases the difficulty of assembly, and a computational evaluation of such tradeoffs would be an interesting analysis possible within this framework.

There can also be other questions of assemblability that our framework could form a foundation to address. As the models get more complicated or more types of joints get involved, additional kinematic geometric interactions could arise. We tackle the validation of the assembly process with low-overhead assembly prototypes generated automatically by our system from the candidate design. We can also use this prototype method to help plan an assembly order. In other words, we provide nonexpert users a scale-down version of the resultant furniture design so that they can instead use low-cost substitutes (e.g., copy paper or acrylic sheet) to create a prototype; thus they can explore its assemblability and assembly plan with this prototype easily and rapidly. By verifying the assemblability of the final flat-pack designs through the paper prototypes, we empower the users to create their own customized furniture [36]. Nonetheless, an algorithm to validate such assemblability and schedule an assemble plan could further advance furniture design for novices [10]; computational algorithms built on our connectivity-based component representation could be used to evaluate such concerns. This ASP problem is extensively researched and explored for engineering structures and machines [5–8]. Specifically, there are several works addressing furniture designs [9, 10], which could be a great enabling technology to extend our work. One of the most promising and efficient methods is to consider the problem from the opposite direction—disassembly: if a furniture represented by a connectivity graph can be separated into individual pieces, then the assemblability can be validated, and the inversion of this disassembly sequence can yield an assembly plan [10].

Our framework enables a basis for computational evaluation systems that can assess or even automate interactions between fabrication, assembly, and performance thus further accelerating the design iteration for novices. Nonetheless, despite the many possibilities for further such automation and computational design, our system—by ensuring the manufacturability of designed furniture models—allows users to much more rapidly iterate designs through the remaining tradeoffs, focusing on achieving their desired mechanical and esthetic functionality.

Lastly, it would be exciting to extend our work to enable active furniture design. Active designs will add reconfigurability to furniture, which allows more interactions with humans in everyday life and could enrich the design space for smart homes [37]. Since our framework is abstract and parameterized, it is straightforward to add new joints and active components to the system. The most challenging part is to define a set of appropriate joints to allow active movements. Thanks to our abstract design scheme, it is possible to replace the stationary planar joints with special active counterparts with controllable actuation (e.g., servo-driven actuation [11]). The active connections could be implemented by the flexible joint, as shown in Fig. 16, which allows angular movements. On the other hand, the ability to codesign electrical components and controlling software for actuation is another challenge, which can refer to the method proposed by Mehta et al. [12]. One example of this active design can refer to Fig. 4(e)–(g).

Ultimately, we have presented an abstract and hierarchical approach as a very general method for design, fabrication, and assembly, which lays the foundation of exploring this class of flat-pack furniture. In the future, we believe this method will enable casual end-users to design, manufacture and assemble various models, such as furniture, architecture, robots, and beyond.

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

Acknowledgments. The authors would like to thank Mr. Christian Warloe and Mr. Gopi Suresh for their valuable comments and helpful suggestions.

Financial support. This work is supported by the National Science Foundation under grant #1644579: A Comptuational Approach to Customizing Design, for which the authors express thanks.

Competing interests. The authors declare that they have no competing interests.

Author contributions. W.Y. and A.M. designed the research, W.Y. and D.Z. conducted the research; W.Y. prepared figures and drafted the manuscript; W.Y., D.Z., and A.M. revised the manuscript; A.M. supervised the research.

References

[1] S.-J. Kim, D.-Y. Lee, G.-P. Jung and K.-J. Cho, "An origami-inspired, self-locking robotic arm that can be folded flat," *Sci. Robot.* **3**(16), 217 (2018).
[2] W. Yan, A. L. Gao, Y. Yu and A. Mehta, "Towards autonomous printable robotics: Design and prototyping of the mechanical logic," **In:** *International Symposium on Experimental Robotics* (Springer, Buenos Aires, Argentina, 2018) pp. 631–644.
[3] W. Yan and A. Mehta, "Towards one-dollar robots: an integrated design and fabrication strategy for electromechanical systems," *Robotica*, 1–17 (2020). https://doi.org/10.1017/S0263574720001101
[4] E. Hawkes, B. An, N. M. Benbernou, H. Tanaka, S. Kim, E. D. Demaine, D. Rus and R. J. Wood, "Programmable matter by folding," *Proc. Natl. Acad. Sci.* **107**(28), 12441–12445 (2010).
[5] F. Demoly, X.-T. Yan, B. Eynard, L. Rivest and S. Gomes, "An assembly oriented design framework for product structure engineering and assembly sequence planning," *Robot. Cim-INT. Manuf.* **27**(1), 33–46 (2011).
[6] C. Sinanoğlu and H. R. Börklü, "An assembly sequence-planning system for mechanical parts using neural network," *Assembly Autom.* **25**(1), 38–52 (2005).
[7] T. Dong, R. Tong, L. Zhang and J. Dong, "A knowledge-based approach to assembly sequence planning," *Int. J. Adv. Manuf. Tech.* **32**(11-12), 1232–1244 (2007).
[8] Y. Wang, Z. Yuan and C. Sun, "Research on assembly sequence planning and optimization of precast concrete buildings," *J. Civ. Eng. Manag.* **24**(2), 106–115 (2018).
[9] W. Pan, Y. Wang and X.-D. Chen, "Domain knowledge based non-linear assembly sequence planning for furniture products," *J. Manuf. Syst.* **49**, 226–244 (2018).
[10] Y. Schwartzburg and M. Pauly, "Fabrication-aware design with intersecting planar pieces," *Comput. Graph. Forum* **32**(2pt3), 317–326 (2013).
[11] A. Mehta, J. DelPreto and D. Rus, "Integrated codesign of printable robots," *J. Mech. Robot.* **7**(2), 021015 (2015).
[12] A. M. Mehta and D. Rus, "An end-to-end system for designing mechanical structures for print-and-fold robots," **In:** *2014 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, Hong Kong, China, May 2014) pp. 1460–1465.
[13] H. Xia, B. Araujo, T. Grossman and D. Wigdor, "Object-oriented drawing," **In:** *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI (ACM, New York, NY, 2016) pp. 4610–4621.
[14] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz and D. Dobkin, "Modeling by example," *ACM Trans. Graph.* **23**(3), 652–663 (2004).
[15] X. Chen, C. Zheng and K. Zhou, "Example-based subspace stress analysis for interactive shape design," *IEEE T Vis. Comput. Gr.* **23**(10), 2314–2327 (2017).
[16] A. Schulz, J. Xu, B. Zhu, C. Zheng, E. Grinspun and W. Matusik, "Interactive design space exploration and optimization for cad models," *ACM Trans. Graph.* **36**(14), 157-1–157-14 (2017).
[17] E. Kalogerakis, S. Chaudhuri, D. Koller and V. Koltun, "A probabilistic model for component-based shape synthesis," *ACM Trans. Graph.* **31**(11), 55-1–55-11 (2012).
[18] S. Chaudhuri and V. Koltun, "Data-driven suggestions for creativity support in 3d modeling," *ACM Trans. Graph.* **29**, 183: (2010).
[19] A. Schulz, A. Shamir, D. I. W. Levin, P. Sitthi-amorn and W. Matusik, "Design and fabrication by example," *ACM Trans. Graph.* **33**(11), 62-1–62-11 (2014).
[20] M. Lau, A. Ohgawara, J. Mitani and T. Igarashi, "Converting 3d furniture models to fabricatable parts and connectors," *ACM Trans. Graph.* **30**(6), 85-1–85-6 (2011).
[21] G. Saul, M. Lau, J. Mitani and T. Igarashi, "Sketchchair: An all-in-one chair design system for end users," **In:** *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI (ACM, New York, NY, 2011) pp. 73–80.
[22] D. Chen, P. Sitthi-amorn, J. T. Lan and W. Matusik, "Computing and fabricating multiplanar models," *Comput. Graph. Forum.* **32**(2pt3), 305–315 (2013).
[23] K. Hildebrand, B. Bickel and M. Alexa, "crdbrd: shape fabrication by sliding planar slices," *Comput. Graph. Forum.* **31**(2pt3), 583–592 (2012).
[24] L. Sass, "Synthesis of design production with integrated digital fabrication," *Automat. Constr.* **16**(3), 298–310 (2007).

[25] M. Claypool, G. Retsin, M. J. Garcia, C. Jaschke and K. Saey, "Combinatorial design: designing collaborative models for construction," *Proceedings of the Convegno Internazionale Naples 2020 International Conference 'DESIGN IN THE DIGITAL AGE'*. (2020).
[26] A. Groenewolt, T. Schwinn, L. Nguyen and A. Menges, "An interactive agent-based framework for materialization-informed architectural design," *Lect. Notes Comput. Sc.* **12**(2), 155–186 (2018).
[27] J. A. Landay, "Technical perspective: design tools for the rest of us," *Commun. ACM* **52**(12), 80–80 (2009).
[28] C. Torres and E. Paulos, "Metamorphe: Designing expressive 3d models for digital fabrication," **In:** *Proceedings of the 2015 ACM SIGCHI Conference on Creativity and Cognition*, C&C '15 (ACM, New York, NY, 2015) pp. 73–82.
[29] N. Umetani, T. Igarashi and N. J. Mitra, "Guided exploration of physically valid shapes for furniture design," *Acm T Graphic* **31**(4), 86-1–86-11 (2012).
[30] A. Schulz, C. Sung, A. Spielberg, W. Zhao, R. Cheng, E. Grinspun, D. Rus and W. Matusik, "Interactive robogami: an end-to-end system for design of robots with ground locomotion," *Int. J. Rob. Res.* **36**(10), 1131–1147 (2017).
[31] K. D. Willis, J. Lin, J. Mitani and T. Igarashi, "Spatial sketch: Bridging between movement & fabrication," **In:** *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '10 (ACM, New York, NY, 2010) pp. 5–12.
[32] Y. Mori and T. Igarashi, "Plushie: an interactive design system for plush toys," **In:** *ACM SIGGRAPH 2007 papers*, (San Diego, CA: USA), ACM (2007) pp. 45-es.
[33] C. Zheng, "Joinery: joints for laser cut assemblies," *Instructables*, (8 Mar. 2018), https://www.instructables.com/id/Joinery-Joints-for-Laser-Cut-Assemblies/.
[34] R. Tian, S. Sterman, E. Chiou, J. Warner and E. Paulos, "Matchsticks: Woodworking through improvisational digital fabrication," **In:** *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, (ACM, New York, NY, 2018) pp. 149-1–149-12.
[35] B. Guy, "How to: design a living hinge" *Ponoko* (31 Jul. 2018). https://www.ponoko.com/blog/how-to-make/how-to-design-a-living-hinge/.
[36] J. I. Lipton, A. Schulz, A. Spielberg, L. Trueba, W. Matusik and D. Rus, "Robot assisted carpentry for mass customization," **In:** *2018 IEEE international conference on robotics and automation (ICRA)* (IEEE, Brisbane, Australia, 2018) pp. 3540–3547.
[37] Wikipedia, "Home automation," *Wikimedia Foundation*, (15 Mar. 2022), https://en.wikipedia.org/wiki/Home_automation.

A. Appendix

A.1. Current Joint Collection

In this section, we introduce the details about three planar joints implemented in our paper.

A.1.1. Finger–finger joint

Finger–finger joints are used to connect two components with their segment of intersection both on the edges. The geometric outline of this finger–finger joint is shown in Fig. 15(a), where alternating rectangular fingers are added to the two coupled edges, and the length of these fingers l_f are matched with the thickness of the material (3 mm in this case) to ensure that the seamless outline of the attached edges. The interval between two fingers forms a dent, which is fulfilled by a finger when the joint is assembled.

As shown in Fig. 15(a), to ensure that the finger–finger joint can indeed hold components together, we introduce a certain amplitude of interference to the joint. In addition, a geometry correction is necessary to compensate for the fabrication kerf introduced by the machines. Thus, when user has specified a interference fit value δ and a certain fabrication method, the convex finger is expanded to the designated value w_f, while the concave dent is trimmed down accordingly. The relationship between these two parameters is described by

$$w_f = w_d + 4\Delta + \delta$$

where Δ is the fabrication kerf and δ is the interference amplitude. For the specific material and fabrication method used, users may need to perform some experiments to find the best interference fit value and the fabrication kerf.

Due to the limitation of 2D fabrication, finger–finger joints are only well suited for connections at angle of 90°. Although user can easily spin components around to form other angles of connections, we do not regard it as the expected usage of finger–finger joint due to the questionable firmness of such connections.

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

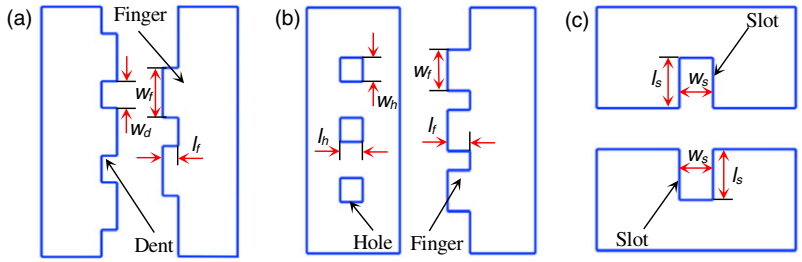

Figure 15. *A collection of planar joints used in our system. (a) Finger–finger joints; (b) Finger–hole joints; (c) Slot–slot joints.*

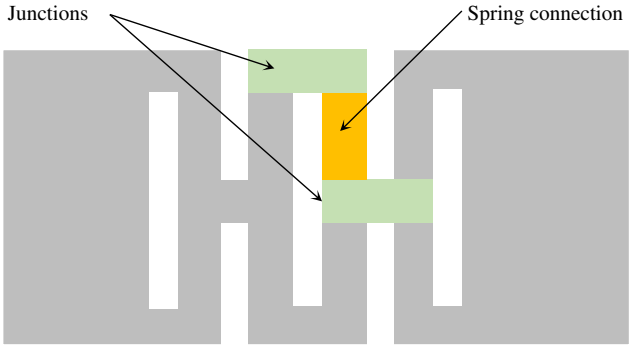

Figure 16. *A basic module of a flexible joint, with its junctions and spring connection structure labeled. The junctions are supposed to behave as rigid joints while the spring connections are responsible for plate bending.*

A.1.2. Finger–hole joint

If the segment of intersection is on the edge of one component while within the face of the other, finger–hole joints are needed. We add fingers to the former component and holes to the latter component (see Fig. 15(b)). Similar to finger–finger joints, users can specify the interference fit value to determine the final geometry of the joint. The relationship between the widths of finger w_f and hole w_h is expressed by

$$w_f = w_h + 4\Delta + \delta$$

where Δ is fabrication kerf and δ is interference amplitude. Again the length l_f of these fingers and length l_h of holes should both be equal to the thickness of the material. Finger–hole joints are also limited to 90° connections.

A.1.3. Slot–slot joint

If the segment of intersection is within the face of both components, we will add a slot–slot joint to connect them. If neither face is fully within each other, then we will cut rectangular slots on both components, and each slot accounts for half of the segment of intersection (see Fig. 15(c)). If one face is fully with another, then we will only cut the slot on one face so that the other face can be stuck through it (see Fig. 11(i)).

The length of the slot l_s is half of the length of the segment of intersection and the width w_s of the slot can be expressed by

$$w_s = w_m + 2\Delta + \delta$$

where w_m is the thickness of material, Δ is the fabrication kerf, and δ is the interference amplitude.

https://doi.org/10.1017/S0263574722000443 Published online by Cambridge University Press

A.2. Flexible Joint

A basic module of flexible joints is shown in Fig. 16. For wider flexible joints, this pattern could be replicated several times along the transverse direction. This module consists of junctions and spring connections. The junctions behave as rigid joints to connect spring connections. The spring junctions function as bending spring to enable angular movement. More details about the design of flexible joints can be found in the reference [35].

A.3. Assembly of Furniture

Table I. *Comparison of assembly of related furniture design systems.*

System	Carpenter skill level	Required tools	Alignment	Time
Ref. [36]	Expert	Mallet, nails, marker	Hard	0–3 h
Ref. [19]	Expert	Mallet, ball joint, hinge joint, screws, etc.	Hard	—
Ref. [29]	Expert	Mallet, nails, marker	Hard	4 h
Our system	Novice	Hammer	Easy	0–30 min

Cite this article: W. Yan, D. Zhao and A. Mehta (2023). "Fabrication-aware design for furniture with planar pieces", Robotica **41**, 48–73. https://doi.org/10.1017/S0263574722000443