FoolProofJoint: Reducing Assembly Errors of Laser Cut 3D Models by Means of Custom Joint Patterns

Keunwoo Park
Hasso Plattner Institute, University of Potsdam, Germany
keunwoo.park@guest.hpi.de

Conrad Lempert
Hasso Plattner Institute, University of Potsdam, Germany
conrad.lempert@hpi.de

Muhammad Abdullah
Hasso Plattner Institute, University of Potsdam, Germany
muhammad.abdullah@hpi.de

Shohei Katakura
Hasso Plattner Institute, University of Potsdam, Germany
shohei.katakura@hpi.de

Jotaro Shigeyama
Hasso Plattner Institute, University of Potsdam, Germany
jotaro.shigeyama@hpi.de

Thijs Roumen
Hasso Plattner Institute, University of Potsdam, Germany
thijs.roumen@hpi.de

Patrick Baudisch
Hasso Plattner Institute, University of Potsdam, Germany
patrick.baudisch@hpi.de

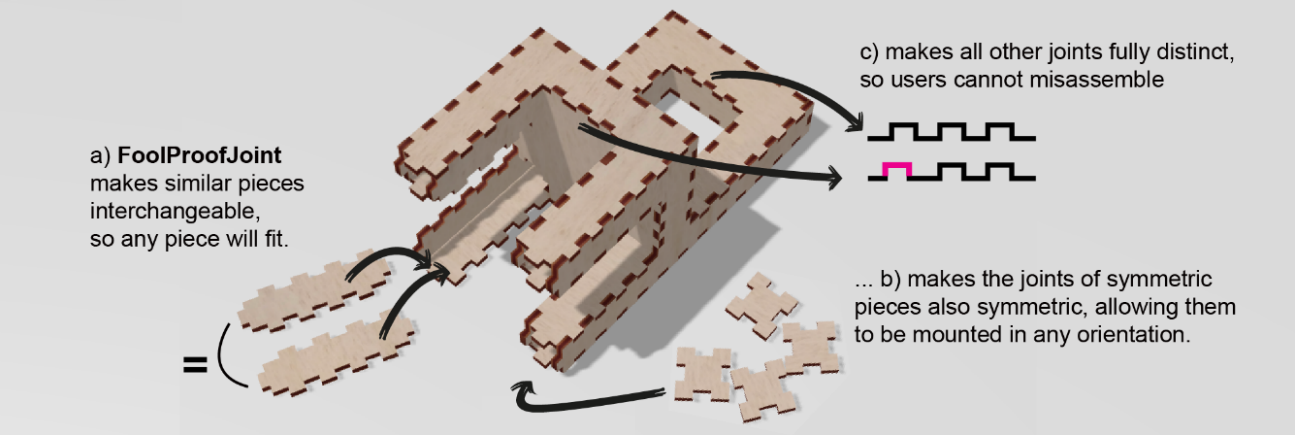

Figure 1: In order to eliminate assembly errors, FoolProofJoint optimizes finger joint patterns, so as to (a) make pieces interchangeable by giving them the same joint patterns where pieces have an identical shape, (b) prevent symmetric pieces from being mounted in the wrong orientation, and (c) prevent incorrect pieces from being assembled.

ABSTRACT

We present FoolProofJoint, a software tool that simplifies the assembly of laser-cut 3D models and reduces the risk of erroneous assembly. FoolProofJoint achieves this by modifying finger joint patterns. Wherever possible, FoolProofJoint makes similar looking pieces fully interchangeable, thereby speeding up the user's visual search for a matching piece. When that is not possible, FoolProofJoint gives finger joints a unique pattern of individual finger placements so as to fit only with the correct piece, thereby preventing erroneous assembly. In our benchmark set of 217 laser-cut 3D models downloaded from kyub.com, FoolProofJoint made groups of similar looking pieces fully interchangeable for 65% of all groups of similar pieces; FoolProofJoint fully prevented assembly mistakes for 97% of all models.

CHI '22, April 29–May 05, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-9157-3/22/04…$15.00
https://doi.org/10.1145/3491102.3501919

CCS CONCEPTS

• **Human-centered computing** → Human computer interaction (HCI); Interactive systems and tools.

KEYWORDS

Personal fabrication, laser cutting, rapid prototyping, manual assembly

ACM Reference Format:
Keunwoo Park, Conrad Lempert, Muhammad Abdullah, Shohei Katakura, Jotaro Shigeyama, Thijs Roumen, and Patrick Baudisch. 2022. FoolProofJoint: Reducing Assembly Errors of Laser Cut 3D Models by Means of Custom Joint Patterns. In *CHI Conference on Human Factors in Computing Systems (CHI '22), April 29–May 05, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3491102.3501919

1 INTRODUCTION

In recent years, laser cutting has become one of the main contenders for fast fabrication. The cutting process itself is fast, it is a subtractive fabrication method which can cut an entire model within minutes [5]. Furthermore, recently developed specialized 3D modeling software, such as FlatFitFab [20], Platener [7], and Kyub [6] allow generating the required 2D cutting plans quickly, while allowing users to model efficiently in 3D.

With fast 3D design and fast fabrication in place, assembly has become the bottleneck that hinders further speed-ups [2]. Assembly requires users to perform a sequence of steps, each of which typically involves the user connecting a piece with another piece based on a pair of matching joints.

Several systems offer methods to facilitate correct assembly, typically with the help of some visual language that communicates which joint to connect to which other joint. FlatFitFab [20] and Kyub [6], for example, engrave matching pairs of numbers; Roadkill [2] instead "points" users to the matching piece directly from within the 2D layout; Daedalus in the dark [10] laser cuts haptic paths in the plates to make such assembly instructions are accessible to blind users.

But assembly instructions have limitations. First, since engraving assembly instructions into the model arguably ruins the appearance of that model, more recent systems cut or engrave next to the pieces [2]. Once a piece is removed from the layout though, piece and instructions are no longer associated, allowing things to go wrong. Second, even with instructions engraved into the pieces, we have observed participants of various laser cutting workshops getting things wrong from time to time, simply due to human error.

In this paper, we address these types of assembly errors using a software tool we call FoolProofJoint. Inspired by techniques used to reduce assembly errors in manufacturing (design for assembly [8]), FoolProofJoint minimizes visual search by making similar pieces fully interchangeable (Figure 1a and b) and prevents assembly errors by making finger joints distinct, unless for pieces that belong together (Figure 1c). In our benchmark set of 217 laser-cut 3D models downloaded from kyub.com, FoolProofJoint made groups of similar looking pieces fully interchangeable for 65% of all groups of similar pieces; FoolProofJoint fully prevented assembly mistakes for 97% of all models.

2 FOOLPROOFJOINT

When users assemble a 3D laser-cut model (Figure 2a), they find themselves looking at a 2D layout of pieces, sometimes referred to as a cutting plan [6], as illustrated by Figure 2b. Users will typically pick up a piece and then look for the matching piece to attach. They may now limit their search to pieces that feature an edge of matching length; In addition, they may have expectations about the overall shape of the desired piece that help them select which piece to pick up.

The assembly situation may thus present itself to the user roughly as illustrated by Figure 2c, where the model's 23 pieces fall into a smaller number of groups based on their rough shape (in the remainder of this paper, we will refer to this shape, i.e., the piece with the finger joints filled in, as envelope). Parts within each group may still differ by their joints—however, users tend to have less clear expectations about what shape of joint they are looking for, as they may find it hard to interpret joint patterns quickly enough to consider them in their visual search.

In this situation, two types of assembly errors tend to emerge. First, users may reach for an incorrect piece (even if it is from that piece group). Second, for pieces with a symmetric envelope, users may try to assemble a piece in the wrong orientation. In the following, we go over these two classes of assembly errors and how FoolProofJoint addresses them.

2.1 Issue 1: Users pick up similar looking pieces

Figure 1a takes a closer look at what happens during assembly. Here the user is trying to assemble one of the chair's legs. The user is looking for a piece that could fit, thus considers all pieces that feature roughly the dimensions of a leg.

Ideally, the user picks the correct piece and successfully assembles it. However, this may not be the most likely outcome, given that there are eight pieces of the same envelope and two more pieces with a similar envelope. There are three other possible outcomes with increasingly dire consequences.

(1) The user picks up an incorrect piece, tries to assemble it, and instantaneously recognizes that it does not fit because their joints collide and cannot be assembled. This case causes little damage, as the user simply drops the piece and tries a different one.

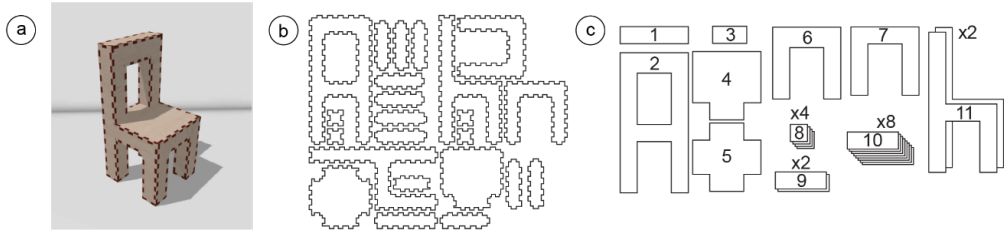

Figure 2: (a) The chair of Figure 1 consists of (b) 23 pieces. (c) However, after filling in their joints we see duplicates, resulting in only 11 groups of pieces.

(2) Less ideally, the incorrect piece still fits. The user attaches it to then now realize that it is incorrect, e.g., because adjacent joints are not aligned. The user removes the piece again and tries a different one. Removing a piece, however, produces a lasting negative effect on both pieces involved, as it wears out their joints, making the final model less sturdy.
(3) The worst possible outcome is illustrated in Figure 3a: an erroneous piece may fit, and the error may go unnoticed, as subsequent pieces continue to fit as well, thereby producing a propagation error. (b) By the time the user recognizes the issue, the user has to dis- and re-assemble *multiple* pieces.

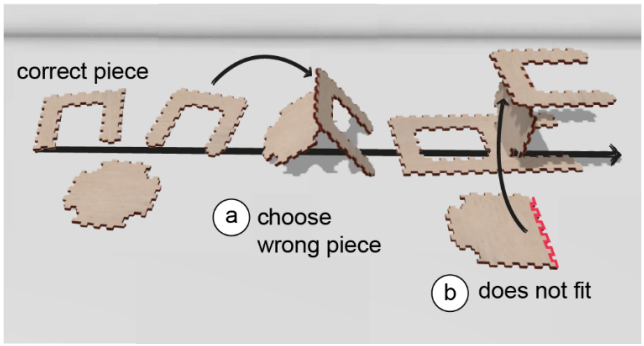

Figure 3: Example of a propagation error: (a) The user has assembled an incorrect piece, but (b) does not find out the mistake until additional pieces have been assembled.

2.1.1 FoolProofJoint makes joints different. FoolProofJoint addresses these assembly mistakes by giving each joint pair a unique finger pattern, which assures that only the correct piece can be assembled. This prevents erroneous assembly and thus also propagation errors by making users aware of their error before any piece can be assembled, giving users the opportunity to put the piece back and try again. Figure 4 shows an example. (a) The user should assemble the *front leg* piece to the *seat bottom* piece, but accidentally assembles the *backrest* piece, which just happens to have a similar envelope. (b) FoolProofJoint resolves this by making the two joint patterns distinct, preventing the incorrect piece from being assembled.

2.1.2 FoolProofJoint makes pieces interchangeable, by making their joints the same. The issue discussed above is caused by pieces *initially* looking the same to the user because they are similar, but then turning out different *later* when the user notices their differences led to assembly mistakes. However, if pieces were the same *all the way*, no problem would ever arise.

FoolProofJoint exploits this idea. Before making any joints different, FoolProofJoint tests whether it might be possible to make pieces fully identical and thus interchangeable. FoolProofJoint achieves this by checking whether the envelopes are the same, whether any potentially present features such as cutouts and engravings are the same, and whether the model geometry offers sufficient degrees of freedom to adjust all involved joints (see Section 5). If that is the case, FoolProofJoint make *the joints* of all pieces which mutually meet these criteria (called a *piece group*) identical.

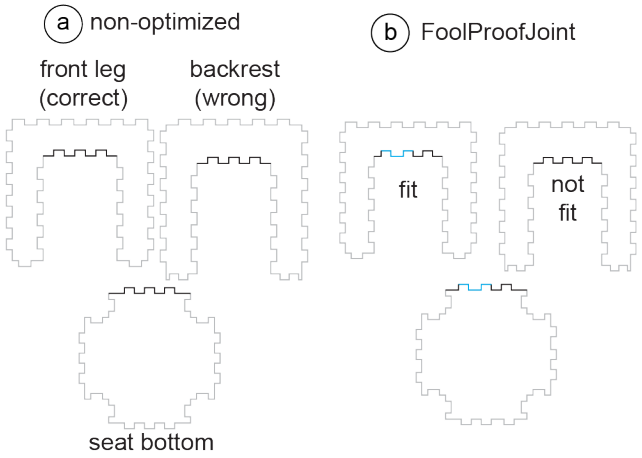

Figure 4: (a) Instead of the correct front leg piece, the user wrongly assembles the backrest piece. (b) FoolProofJoint prevents this by making joint patterns distinct.

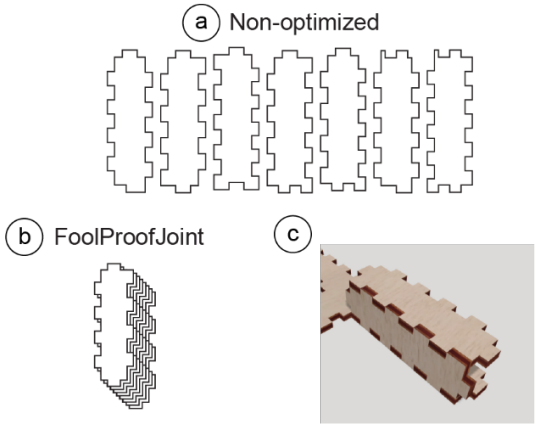

Figure 5: The chair example features eight leg pieces with the same envelope. (a) Non-optimized pieces can have different joint patterns. (b) By making their joint patterns the same, FoolProofJoint makes these pieces interchangeable, allowing users to assemble *any* of them at any fitting connection, thus reducing visual search. (c) The interchangeable pieces of legs are connected to each other by rotating 180°which is necessary to completely fill their shared edge.

Figure 5 shows an example. Here FoolProofJoint makes the joints of all eight leg pieces of the chair model identical, thus the pieces are fully interchangeable. This not only eliminates the risk of assembly errors, but also vastly reduces the visual search to find correct pieces.

2.2 Issue 2: Users accidentally re-orient symmetric pieces

The second issue that leads to assembly errors are symmetric piece envelopes. Looking back at Figure 2, shows that all pieces except for the ones in group 11 feature some sort of symmetry; several pieces even feature multiple symmetries. As a result, users might

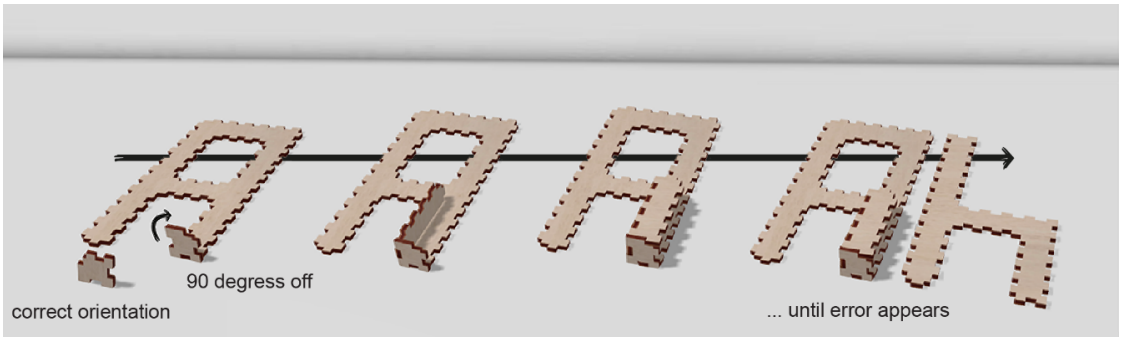

Figure 6: The user assembles the square foot piece in the wrong orientation causing a propagation error which makes it impossible to insert the side piece a few assembly steps later.

try to assemble e.g., the chair's feet in any of the four possible orientations. As a result, we obtain the same range of outcomes, i.e., the user may attach the piece in the correct orientation or—the user may try an incorrect orientation, causing the piece either not to fit, or to fit despite the incorrect orientation as illustrated by Figure 6, which leads to propagation errors.

In our experience, issues resulting from symmetrical piece envelopes even happen when users are provided with an instruction manual: Assembly instructions will often do a good job at guiding users to the right piece and even the right orientation. However, once users have picked up that piece, the piece's original orientation is quickly lost, as the piece gets rotated or flipped in the user's hand.

2.2.1 Solution: FoolProofJoint makes symmetric joints. FoolProofJoint addresses issues resulting from symmetric piece envelopes in a similar way to how it addresses confusion of pieces. FoolProofJoint starts by testing whether it might be possible to make the joints of the piece symmetric in a way that matches its symmetries. FoolProofJoint achieves this by checking whether the piece is symmetric, whether no features such as cutouts or engravings are present that might break symmetries, and whether the model geometry offers sufficient degrees of freedom to adjust all involved joints (see Section 5: 'Algorithm'). If that is the case, FoolProofJoint makes the piece's joints identical according to the piece's symmetry (Figure 7).

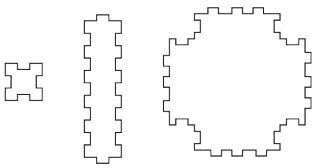

Figure 7: Examples of rotationally symmetric pieces. If possible, FoolProofJoint makes the joints of pieces symmetric to give the piece the same symmetries as its envelope.

3 CONTRIBUTION

In this paper, we make four contributions. First, we categorize and describe common classes of mistakes users make when assembling laser-cut 3D models, namely those stemming from similar and symmetric pieces. Second, we demonstrate how to resolve these issues by making pieces interchangeable when possible and distinct otherwise. Third, we present an algorithm that maximizes the effect of our approach by means of global optimization. Finally, we validate our approach by means of a technical evaluation involving 217 laser-cut 3D models, where FoolProofJoint made groups of similar looking pieces fully interchangeable for 65% of all groups of similar pieces and fully prevented assembly mistakes for 97% of all models.

The benefit of our approach is faster, more reliable assembly. Limitations include that our current implementation is limited to finger joints.

4 RELATED WORK

Our work builds on related work in laser cutting within personal fabrication, computational joint design, assembly assistance and design for assembly.

4.1 Laser cutting in personal fabrication

Laser cutting is an efficient and high precision manufacturing technique in personal fabrication. Unlike other free form manufacturing devices, such as 3D printers and CNC milling machines, laser cutters can only cut 2D polygonal shapes from planar materials. However, connecting 2D laser-cut pieces can turn planar designs into 3D shapes. For example, one can approximate 3D shapes with cross-sectional cuts [16], wireframe-like pieces [12] and shell assembly [11]. LaserOrigami [21] enables creating 3D shapes by cutting and bending acrylic plates in a unified process with the help of a defocused laser.

To allows novice users to design 3D solid models from scratch that are inherently laser-cuttable, Kyub [6] proposes a boxel-based interactive design system that takes care of all the underlying necessities for cutting and assembling the model, such as finger-joint arrangement, cutting and assembly instructions. For functional purposes, researchers presented various integrated design and optimization systems. FlatFitFab [20] is a sketch-based laser-cut modeling system, which also simulates and optimizes the physical stability of cross-sectional models. For 3D solid models, FastForce [1] provides a reinforcement tool that optimizes the assembly configurations to enhance the durability of 3D laser-cut models. Benefitting from the flexibility of laser-cutting, people add more high-level functionalities to laser-cut models, such as joints and mounts [27], kerf-canceling connections [26], or even fully functional mechanisms [23].

In most of the laser-cutting design and assembly systems, planar laser-cut pieces are connected with custom-designed joints. The quality of joint design influences stability, assemblability and aesthetics of the final model. FoolProofJoint helps creating such finger-joint connections that prevent assembly mistakes.

4.2 Computational joint design

Creating assemblies with computer generated joints has been researched in numerous fields. According to Wang et al., [30] the main objectives for generating joints include parts connectivity [13], part mobility [33] and joint strength [34].

Procedural generation of such connections has been demonstrated by Whiting et al. [32] for structurally sound masonry buildings. Matchsticks [28] introduces CNC milled mortise-and-tenon connections, while computationally generated 3D printed joints have been used in TrussFab [17] and Magrisso et al. [19]. Yao et al. [34] have explored computationally generating decorative inserts. Joinery [35] is a software tool that generates joint patterns for user selected edges.

In this paper we focus on finger-joints that are carved out of the of the parts which are intended to connect. Traditionally these are generated as repetitive patterns, however in this work we propose an algorithm to make these connections unique for preventing mis-assembly.

4.3 Assembly assistance

A common way to assist assembly is to use static and procedural manuals. However, according to Parmentier et al. [24], procedural manuals have the following problems: First, users often do not use instructions. Second, looking at instructions and interpreting them imposes a cognitive load on users. Third, they have a negative effect on motivation. Therefore, many research projects have proposed more advanced methods to instruct assembly processes to users.

A direct consideration is enhancing the visual procedural manuals with more intuitive assembly sequences and informative visualizations [3]. For example, Agrawala et al. [4] present a system for generating effective assembly instructions which satisfy basic building principles, and Yan [31] extends the static paper-based manual with real-time overlays in augmented reality. Moreover, extra physical add-ons can help to guide the assembly process by pre-connecting the assembly components, such as 3D printing connected tiles [11], rigid bricks with latex layers [14], or elastic trusses with pre-stretched fabric [25]. Roadkill [2] guides users in assembling 3D laser cut models by arranging the connecting parts in an intuitive planar layout.

4.4 Design for assembly

In industrial manufacturing, accurate and quick assembly is an essential factor for productivity. If humans assemble manufactured parts, product designers need to design the parts so that they are easy to assemble. This assembly-aware design is known as Design for Assembly, or DFA for short [9].

Manufacturing engineers, designers, and researchers established general guidelines for DFA from accumulated cases and studies. For example, Bougue [8] and Lu et al. [18] provide comprehensive general guidelines for DFA, also highlighting the importance of a mistake-proof design that eliminates systematic errors in human assembly.

FoolProofJoint reduces assembly mistakes by translating traditional general DFA guidelines to laser-cut fabrication. It implements three of the key rules of DFA, namely (1) aiming for mistake-proof designs [8], (2) minimizing connector types to simplify the design and ease of the assembly process [18], and (3) designing for simple part orientation [8].

5 ALGORITHM: HOW FOOLPROOFJOINT DESIGNS FINGER JOINTS

The primary objective of FoolProofJoint is to make pieces and their orientations interchangeable to eliminate assembly errors. As we will see, for some model geometries, however, the problem is overconstrained, forcing FoolProofJoint to resolve the issue by instead making the joints of a subset of pieces distinct, rather than identical. Algorithm 1 shows this procedure in pseudo code.

5.1 Corners

The FoolProofJoint algorithm designs joints as an alternating sequence of fingers and gaps, each of which has a certain width. It considers finger joint patterns as being composed of three parts: *beginning*, *mid-section*, and *end*. FoolProofJoint starts by determining the *beginning*, i.e., whether the joint pattern at hand needs to start with a finger or a gap and similarly whether the *end* will be a finger or a gap. This also determines whether the total number of fingers and gaps is even or odd.

To determine *beginning* and *end*, FoolProofJoint looks at the model in its entirety. The reason is that the pieces of a model interact with other pieces at these points. As illustrated by Figure 8a, the corners of a cube or other rectilinear geometry is where *three* pieces come together. As we can see, only one piece can extend all the way into the corner—we call this the *primary corner*. Then, a second piece touches the primary corner; we will refer to this corner as the *secondary corner*. The remaining space is filled by what we will call the *tertiary corner*. (b) When there are more than three pieces meeting at a corner on e.g., an octahedron, there is even a *quaternary corner*.

Beginning and *ends* of a joint thus assume one of the possible corners, e.g., they can form a primary, secondary, or tertiary corner when three pieces are meeting at a corner.

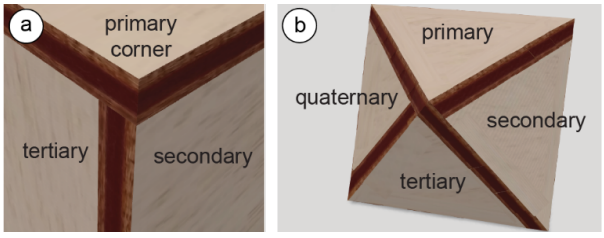

Figure 8: (a) Designing the corner of rectilinear geometry requires deciding which of the three adjacent pieces is *primary*, *secondary*, and *tertiary*. (b) Non-rectilinear models can feature additional corner types. This octahedron, for example, also features *quaternary* corners.

Algorithm 1 FoolProofJoint

```
Input: model
Output: finger joint patterns for every edge
// the corners, i.e., beginning and end of joints are determined (see sections 5.4 to 5.7)
corner_assignment = assign_corners(model)
// the middle parts of the joints are determined to prevent mis-assembly (see sections 5.8 to 5.10)
mid_section_joints = generate_mid_section_joints (model, corner_assignment)
// together they form the finger joints of the model
return corner_assignment, mid_section_joints
assign_corners:
Input: model
Output: corner assignment for every corner
// corner types are primary, secondary, tertiary (in some cases more, see Figure 8b)
for the corner types in the model
   // Section 5.4
find interchangeable pieces and orientations for each piece
   // Section 5.5
   create corner variables and set priorities
   sort corner variables by priority
   for each corner variable
      assign corner to the variable if its preferred value is 1
      // Section 5.6
   update preferred values of the other variables adjacent to this variable
// Section 5.7
handle joints connecting pieces with the same envelope
return corner assignment
generate_mid_section_joints:
Input: model, complete corner assignment
Output: finger joints for every edge
// Section 5.8
generate initial joints and find ambiguous joint pairs
while ambiguous joint pairs exist
   find the joint with the most ambiguous joints
   // Section 5.9
   find other joints to modify together with the found joint
   // Section 5.10
   modify the joints
return joints
```

5.2 Interchangeability is an optimization problem

As we stated earlier, it is not always possible to make all pieces with the same envelope interchangeable. Figure 9 proves this using a counter example. A cube consists of six pieces with identical envelopes. Thus, the optimal solution would be to create the cube from six interchangeable pieces. However, since a cube features *eight* corners, all of which feature exactly one primary corner it's not possible to distribute these eight primary corners to its six pieces evenly. If the pieces don't have the same number of primary corners, there is no way to make them all interchangeable.

Because these geometries exist, FoolProofJoint computes interchangeability as an optimization process that tries to minimize the number of types of pieces and tries to maximize rotational symmetricity. While it is often impossible to create a perfect solution,

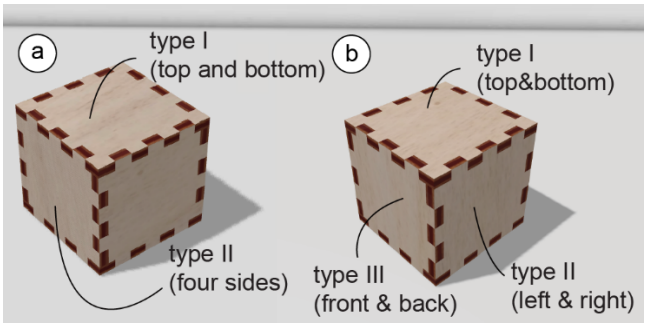

Figure 9: Cubes feature six sides, but eight corners, (a) a solution requires at least two types of pieces. (b) An alternative solution with two-fold symmetric pieces requires three types of pieces.

FoolProofJoint still produces highly useful results, e.g., Figure 9a for the cube.

Finding the solution which allows for the most interchangeability by brute forcing is not computationally feasible. (Example: the chair model has 42 corners; each corner allows for six possible configurations of primary and secondary corners, resulting in a search space of 10^{32}corner configurations). FoolProofJoint therefore uses a heuristic algorithm, as presented in the next section.

5.3 Outline of the FoolProofJoint algorithm

FoolProofJoint assigns corners with the objective of making pieces and orientations interchangeable. Once corners have been assigned, FoolProofJoint fills mid-sections while preventing mis-assemblies.

The corner assignment consists of two steps. First, FoolProofJoint assigns priorities for the pieces intersecting at each corner. Second, it decides whether pieces should or should not fill corners to achieve interchangeability and symmetricity.

To perform this computation, FoolProofJoint uses three main variables per piece and corner. *Corner variables* are set to 1 if the piece fills this corner. *Priorities* indicate the order in which FoolProofJoint assigns values to the corner variables. *Preferred values* represent what value should be assigned to a *corner variable* to fulfill interchangeability and symmetricity.

5.4 Finding interchangeable pieces and orientations

As a preprocessing step, FoolProofJoint identifies interchangeable pieces by comparing their envelopes. If the pieces have the same envelope, then FoolProofJoint tries to make them interchangeable. FoolProofJoint detects pieces with the same envelopes by characterizing the pieces by their lengths and internal angles. If two pieces have the exact same edge lengths and angles in the same order, we decide that they have the same envelope.

FoolProofJoint finds a piece's rotationally symmetric orientations by rotating the piece and checking if the rotated envelopes match with the original envelope. Multiple rotations can be identical. We treat every corner angle of the piece as a potential angle for rotational symmetry. If pieces have textures or cutouts, we assume they cannot be rotated, because textures and cutouts might not be rotationally symmetric.

5.5 Set priorities

FoolProofJoint assigns high *priorities* to pieces from groups containing many pieces (such as the chair's legs) and to pieces with many symmetries (such as the chair's feet). The reason is that as FoolProofJoint assigns values to the corner variables, it becomes harder to fulfill interchangeability and symmetricity, because of constrains imposed by the corner variables which were already assigned. If FoolProofJoint must break interchangeability and symmetricity, it will do so for the corners with a lower priority value. Or specifically, FoolProofJoint initializes *priority* with the number of *corners* that need to be made identical to achieve interchangeability and symmetricity.

Figure 10a shows an example, in which FoolProofJoint is trying to make two pieces A and B interchangeable and rotationally symmetric. To make A and B interchangeable, the corner variables $p_{A,\ 1}$ and $p_{B,1}$ that describe the "top left" corner on each piece must be the same. Considering the rotational symmetry of A and B, an optimal solution would fulfill the equation $p_{A,1} = p_{A,2} = p_{B,1} = p_{B,2}$. As a result, FoolProofJoint assigns all four corners a priority of 3.

5.6 Update preferred values

A piece may not be allowed to fill a corner, because that corner is already occupied by another piece. The two pieces in Figure 10b, for example, meet at an edge. FoolProofJoint reflects this by updating the *preferred values* of pieces touching at a corner. Also, FoolProofJoint updates preferred values of corners on the other side of the edge to satisfy symmetricity.

When no piece wants to fill a corner, then the corner is assigned to the piece with the with the lowest priority at that corner by setting its *preferred value* to one. Initially, preferred values are set to 1.

FoolProofJoint provides a special treatment to concave edges, such as the one shown in Figure 11a. For this type of geometry, sliding in a horizontal piece with corners can be difficult, as these tend to interlock with the fingers in the sides. FoolProofJoint therefore locates concave edges first and to assign *corner variables* as 1 as to favor the type of corner placement shown in Figure 11b. FoolProofJoint proceeds by assigning corners as described above.

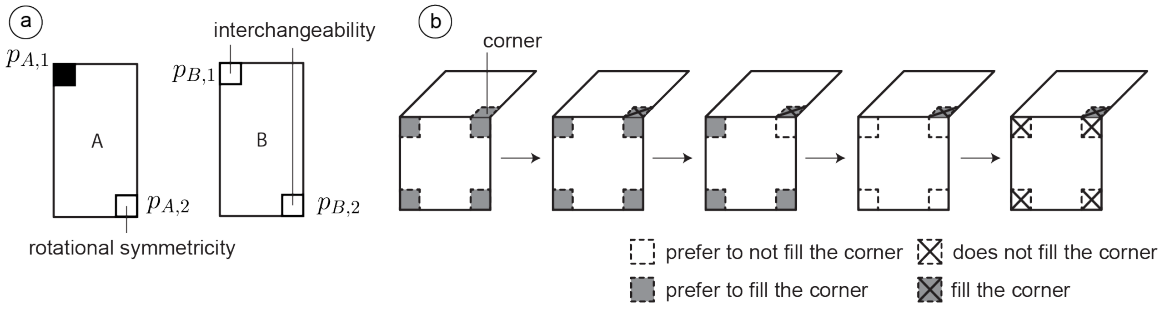

Figure 10: (a) FoolProofJoint determines that the corner variables should be equal to achieve interchangeability and symmetricity. (b) FoolProofJoint updates preferred values of touching pieces along with the corner variable value. In this case, preferred values are set to 0 to make the vertical piece rotationally symmetrical.

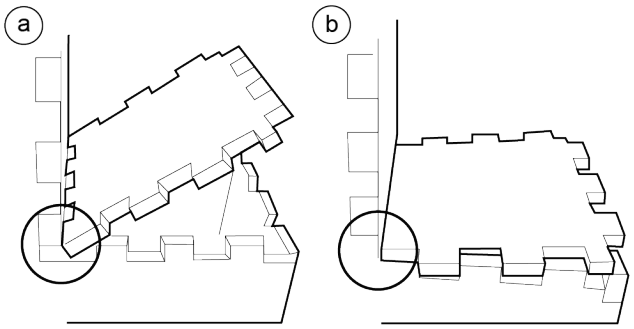

Figure 11: a) The piece with concave corner in its outline does not fill the corner. Therefore, users have to slide a piece between already existing fingers. b) By filling concave corners, it is easier to assemble pieces to concave corners.

5.7 Joints connecting pieces with the same envelope

If pieces of the same envelope are connected (as is the case in Figure 10a), FoolProofJoint postpones assigning their corners. FoolProofJoint assigns the corner to the third piece participating in that corner (by setting its *preferred value* to 1, the others to 0), as assigning it to either of the two candidates would break interchangeability on the spot.

If the third piece's *preferred values* happen to be zero as well, FoolProofJoint creates rotationally interchangeable pieces. The chair legs shown in Figure 5c, for example, fit together if one leg is rotated by 180°. FoolProofJoint produces this type of interchangeability by assigning one corner to each of the interconnected joints.

If it is not possible to make the pieces fill the equal number of corners, FoolProofJoint fills corners with as few pieces as possible. The cube shown in Figure 9, for example, consists of six pieces. Since, as discussed in Section 5.2, FoolProofJoint cannot assign eight corners to six interchangeable pieces, FoolProofJoint lets the top and bottom pieces fill the corners to minimize the number of pieces that fill primary corners as shown in Figure 9a. Then the other four pieces can be made interchangeable. After filling the primary corners eight secondary corners are left. This time, four pieces can have secondary corners. Therefore, each piece gets two secondary corners. Besides this solution, there exists another solution which creates only mirror symmetric pieces as shown in Figure 9b. This solution does not equally distribute secondary corners, so it creates three different pieces.

5.8 Generating joints, preventing assembly errors

Given that *beginning* and *end* of each joint have been defined, it comes down to filling the *mid-section*. If *beginning* and *end* have been assigned primary or secondary corners, or if neither has been assigned primary or secondary corners, the total number of fingers and gaps is odd. FoolProofJoint aims to fill mid-sections uniformly with finger widths of twice the material thickness of the pieces, as suggested by Hasluck [15]. FoolProofJoint thus sets the total number of fingers and gaps in a joint to the nearest even or odd number to $\frac{length\ of\ edge}{optimal\ finger\ width}$.

Next, FoolProofJoint adjusts finger joints to prevent assembly errors. FoolProofJoint starts by identifying pairs of joint patterns that fit but form an incorrect connection. In the following, we call these joints *ambiguous joints*.

FoolProofJoint finds ambiguous joint patterns in three steps, as shown in Figure 12. FoolProofJoint searches for ambiguous joints on the target piece in Figure 12. The model has a single correct mating joint. First, FoolProofJoint finds pieces that have the same joint pattern and their counterparts. Second, FoolProofJoint checks if connecting the found pieces and the target piece is valid or not by looking at the connected pairs in the model. In the last step, FoolProofJoint checks if the invalid pieces from the previous steps are similar to the correct counter piece. This step filters out pieces with a dissimilar envelope that are unlikely to be confused. Users are unlikely to confuse e.g., B with the correct piece. However, A has a very similar envelope and might be confused. Thus, the piece is considered ambiguous, as well as the joint pattern which fits to the target piece.

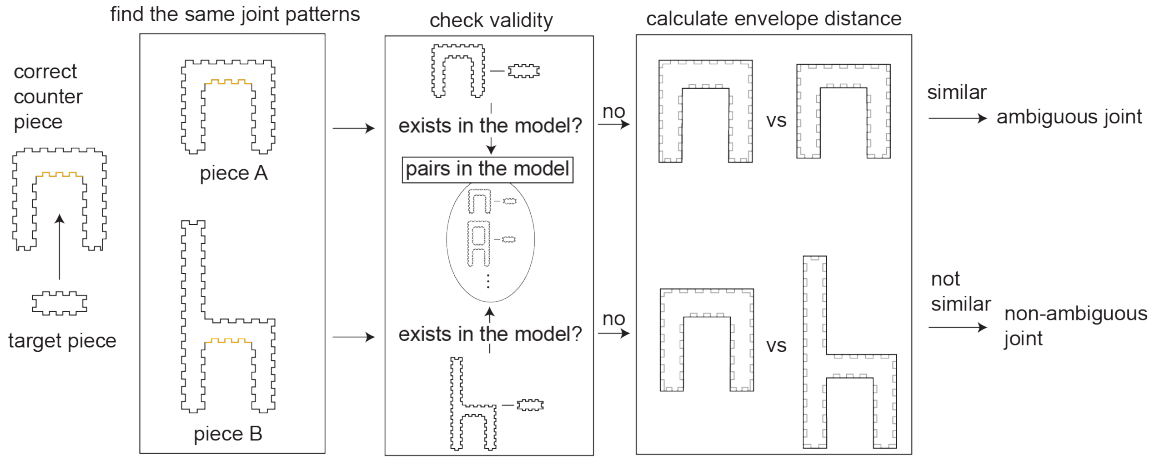

Figure 12: Ambiguous joint patterns finding process.

FoolProofJoint determines the similarity of two pieces by calculating a distance between the shapes of their envelopes. It considers envelopes as similar if this distance is below a threshold which we determined by analyzing the models from our test set. It uses the turning function [20, 22, 29] to calculate distances between the shapes of envelopes. A turning function represents a shape as a graph of normalized length and angle of each point, as shown in Figure 13a. The difference between two shapes can be calculated by integrating the difference between their two turning functions. However, turning functions are scale invariant. To take scale into account, we multiply the turning function difference between the shapes by the ratio of their areas. As a result, the distance between two envelopes is computed as $turning\ function\ difference * (\frac{bigger\ envelop\ area}{smaller\ envelop\ area})$.

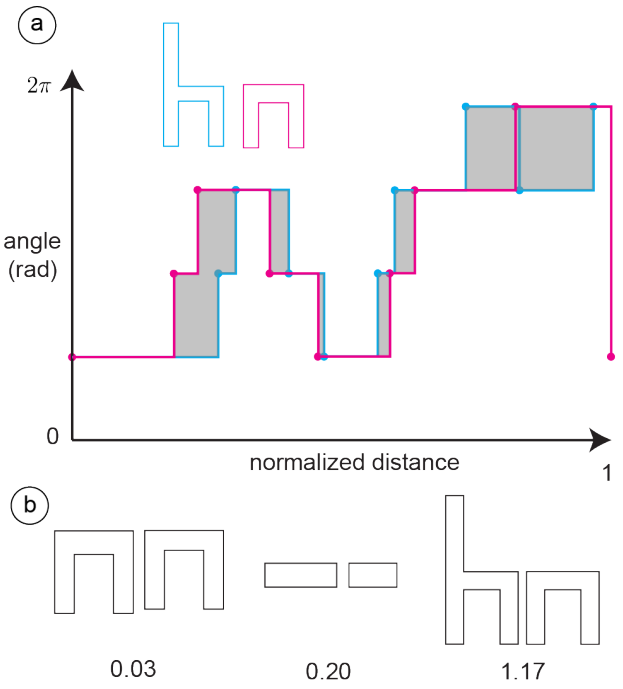

Figure 13: a) Turning functions of two sample pieces from the chair model. Gray area is the difference of the turning functions. b) Distances of envelopes in the chair model.

To make the distance value rotationally invariant, FoolProofJoint creates turning functions in all possible orientations and calculates differences for all pairs of two envelopes and chooses the smallest one. We considered two envelopes are similar if their distance is smaller than 0.1. Figure 13b shows distance examples.

5.9 Determining which joint patterns should be the same or opposite

Updating a single joint pattern can break interchangeability between pieces and the piece's symmetricity. Therefore, FoolProofJoint finds joint patterns that need to be updated together. First, it looks for patterns that should be the same, then, it finds the mating joints. For example, as shown in Figure 14a, there are two ambiguous joints highlighted and FoolProofJoint tries to update joints in the gray pieces. FoolProofJoint needs to change one of the ambiguous joints to prevent mis-assembly. Therefore, FoolProofJoint marks one of the joint patterns to be updated and it updates the mating joint accordingly. It repeats this for all other joint patterns that connect to interchangeability and symmetricity.

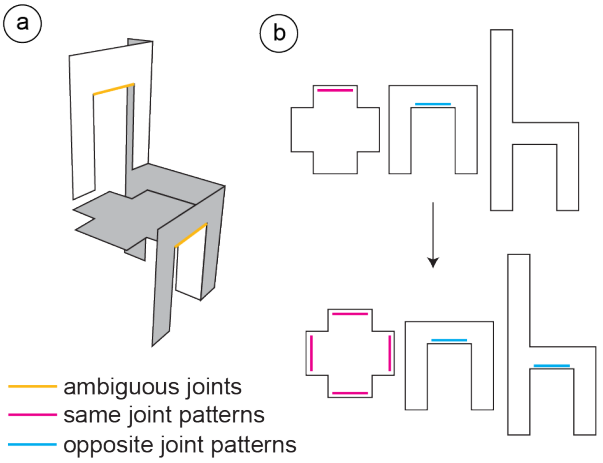

Figure 14: FoolProofJoint updates multiple joint patterns together to keep interchangeablity and symmetricity.

However, if a piece is in the middle of multiple ambiguous joints, it is not possible to maintain the interchangeability and symmetricity. For example, the model in Figure 15 has two ambiguous joints: J1 and J2. Therefore, J1 and J3 should be changed, because A is rotationally symmetric. The best solution is to only break the symmetricity of B.

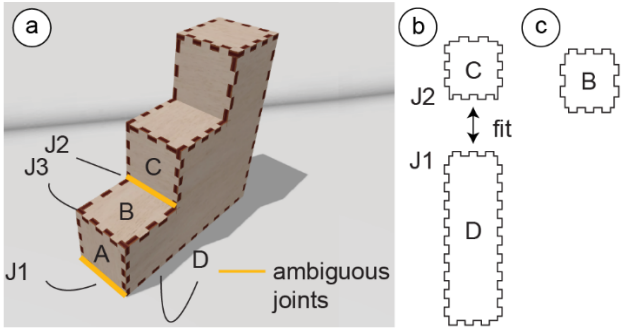

Figure 15: a) An example case of breaking symmetricity for disambiguating joints. b) C and D have ambiguous joint patterns, i.e., connecting them is incorrect even though it is possible. c) B is rotationally symmetric, before modifying the ambiguous joint.

5.10 Disambiguating joint patterns

FoolProofJoint differentiates joint patterns by shifting fingers. We represent these shifts as digits of a ternary numerical system. If the

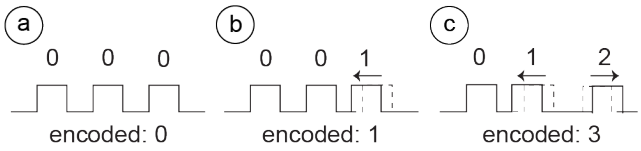

Figure 16: a) Either fingers or gaps are considered as bits of a ternary number. When number 0 is encoded, nothing changes. b) When 1 is assigned to a finger or gap, it shifts to left. c) When 2 is assigned, fingers or gaps shift to right.

digit is 0, the corresponding finger is not shifted. If it is 1 as shown in Figure 16b, the corresponding finger is shifted to the left. When it is 2, it is shifted to the right. This results in $3^{\#fingers}$ possible encodings for a joint pattern of a given length. If this is not enough to prevent all assembly mistakes, FoolProofJoint adds more fingers if the finger width is not less than 4 mm. FoolProofJoint shifts in intervals of 2 mm. This difference is large enough to prevent assembly of incorrect joint patterns and small enough to not interfere with neighboring fingers.

FoolProofJoint finds ambiguous joint pairs in a model. It changes the pattern of the most frequent joint in that collection and adjusts mating candidates to be either all the same or unique. FoolProofJoint repeats this until all joints are disambiguated.

6 TECHNICAL EVALUATION

We conducted a technical evaluation, in which we downloaded 3D laser-cut models from an online repository and applied the FoolProofJoint algorithm to create finger joint patterns. We evaluated to what extent FoolProofJoint would make similar pieces interchangeable, symmetric pieces fully rotationally symmetric, and disambiguate all other joints.

6.1 Metrics

We evaluated the success of the FoolProofJoint algorithm in terms of three metrics. First, we evaluated the degree of piece interchangeability across groups of pieces with the same envelope. We examined whether *all* pieces in these groups had been made fully interchangeable. Second, we evaluated how many pieces with a rotationally symmetric envelope were symmetric after assigning joint patterns. Third, we evaluated the prevention of assembly mistakes by evaluating whether any wrong connections could be made in each model.

6.2 Data set

We downloaded the top 104 hits returned by ranking models on kyub.com by popularity, as shown in Figure 17. Many of them contained multiple models, resulting in an overall number of 220 models. We removed the three models that did not contain any finger joints, leaving us with 217 models, which we used in our evaluation.

We applied the FoolProofJoint algorithm to all 217 models. Figure 18 shows four example results.

6.3 Results

Figure 19 summarizes our findings. For 65% of groups of pieces with the same envelope, FoolProofJoint succeeded at making *all* pieces in the group interchangeable. The remaining groups were still successfully processed but resulted in more than one set. FoolProofJoint made 47% of the pieces the envelope of which featured some sort of symmetry fully symmetric by providing them with matching joint patterns. 7% of the groups that ended up non-interchangeable had been made interchangeable in the first steps of FoolProofJoint, yet FoolProofJoint then sacrificed this interchangeability later, to guarantee disambiguation (Section 5.9).

For 211 out of 217 models (97%), FoolProofJoint produced a result that cannot be mis-assembled, as *all* pieces had either been

Figure 17: We downloaded 104 search results, excluding 3 results without finger joints, and extracted 217 models.

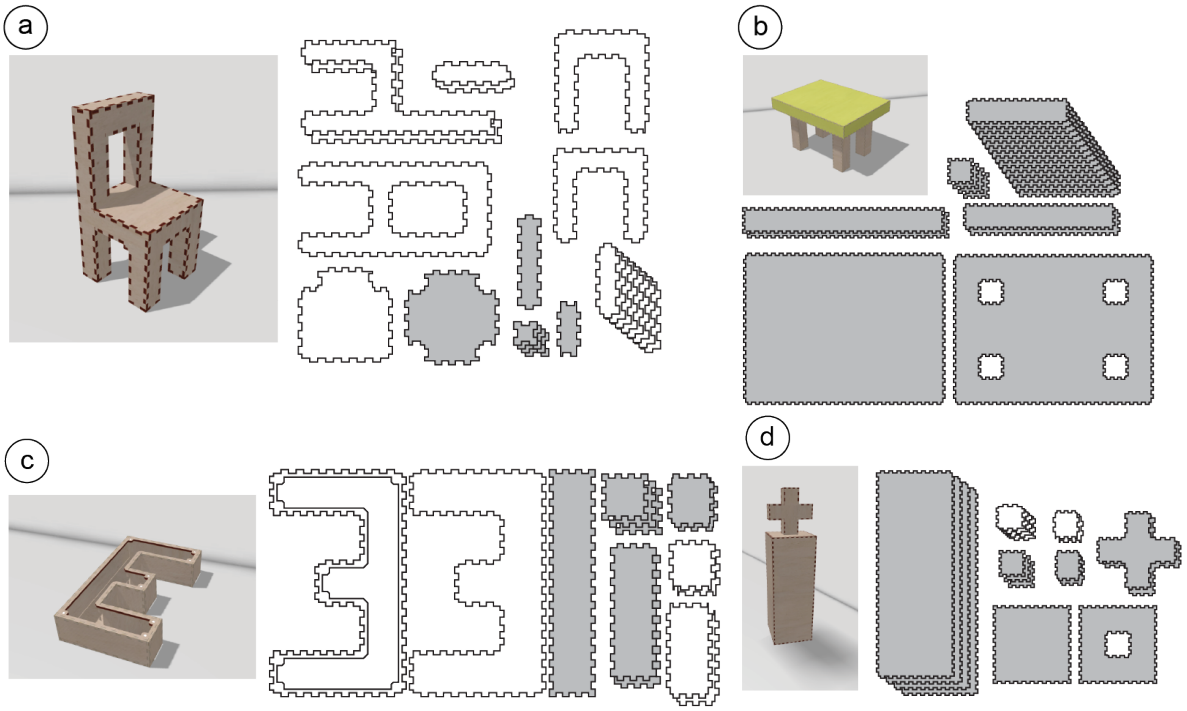

Figure 18: Example results from the dataset. In the 2D view, the stacked pieces are interchangeable, and the gray pieces are rotationally symmetric.

made interchangeable or have been disambiguated. The 6 remaining models contained edges that were too short to allow FoolProofJoint to create varied joint patterns necessary for disambiguation.

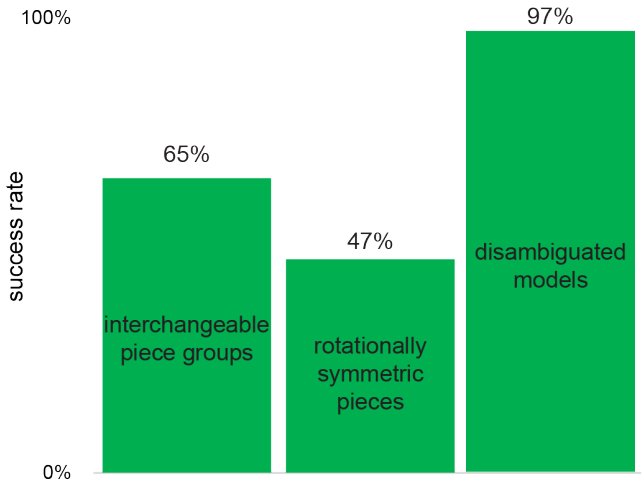

Figure 19: Results on interchangeability, rotational symmetry, and prevention of assembly mistakes

6.4 Discussion

Our results show that in 97% of cases, *all* joints of the resulting models had been successfully disambiguated, so that users will not be able to connect wrong pieces.

For most models, FoolProofJoint went a step further and made groups of parts interchangeable and/or symmetric parts fully symmetric. This simplifies assembly as discussed throughout this paper, in that this allows users to now pick up *any* piece that features the right envelope and/or allows users to attach parts in any orientation.

These insights combined suggest that FoolProofJoint indeed succeeds at simplifying assembly and reducing the potential for assembly errors.

7 CONCLUSION

We have presented FoolProofJoint, a software tool that adjusts finger joint patterns to reduce the likelihood of assembly errors by (1) giving similar pieces identical joint patterns to make them interchangeable, (2) giving pieces symmetric joint patterns according to their symmetry, and (3) giving everything else distinct joint patterns.

FoolProofJoint is as a step towards fast and most of all, *reliable* assembly. This step matters, as fast cutting machines and efficient design tools have sped up rapid prototyping, with assembly being the new bottleneck. FoolProofJoint also makes assembly more predictable. By eliminating the particularly time-intensive types of errors, such as propagation errors, FoolProofJoint has the potential to help *groups of users* finish their assemblies roughly at the same time. This matters when running workshops and in school class, where predictable timing is of the essence.

As future work, we plan to extend our approach to additional joint types and explore other ways of making assembly more predictable.

REFERENCES

[1] Muhammad Abdullah, Martin Taraz, Yannis Kommana, Shohei Katakura, Robert Kovacs, Jotaro Shigeyama, Thijs Roumen, and Patrick Baudisch. 2021. FastForce: Real-Time Reinforcement of Laser-Cut Structures. Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, Article 673, 1–12. DOI:https://doi.org/10.1145/3411764.3445466

[2] Muhammad Abdullah, Romeo Sommerfeld, Laurenz Seidel, Jonas Noack, Ran Zhang, Thijs Roumen, and Patrick Baudisch. 2021. Roadkill: Nesting Laser-Cut Objects for Fast Assembly. In The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21). Association for Computing Machinery, New York, NY, USA, 972–984. DOI:https://doi.org/10.1145/3472749.3474799

[3] Maneesh Agrawala, Wilmot Li, and Floraine Berthouzoz. 2011. Design principles for visual communication. Commun. ACM 54, 4 (April 2011), 60–69. DOI:https://doi.org/10.1145/1924421.1924439

[4] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. 2003. Designing effective step-by-step assembly instructions. ACM Trans. Graph. 22, 3 (July 2003), 828–837. DOI:https://doi.org/10.1145/882262.882352

[5] Patrick Baudisch and Stefanie Mueller (2017), "Personal Fabrication", Foundations and Trends®in Human–Computer Interaction: Vol. 10: No. 3–4, pp 165-293. http://dx.doi.org/10.1561/1100000055

[6] Patrick Baudisch, Arthur Silber, Yannis Kommana, Milan Gruner, Ludwig Wall, Kevin Reuss, Lukas Heilman, Robert Kovacs, Daniel Rechlitz, and Thijs Roumen. 2019. Kyub: A 3D Editor for Modeling Sturdy Laser-Cut Objects. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, Paper 566, 1–12. DOI:https://doi.org/10.1145/3290605.3300796

[7] Dustin Beyer, Serafima Gurevich, Stefanie Mueller, Hsiang-Ting Chen, and Patrick Baudisch. 2015. Platener: Low-Fidelity Fabrication of 3D Objects by Substituting 3D Print with Laser-Cut Plates. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15). Association for Computing Machinery, New York, NY, USA, 1799–1806. DOI:https://doi.org/10.1145/2702123.2702225

[8] Robert Bogue. 2012. Design for manufacture and assembly: background, capabilities and applications, Assembly Automation, Vol. 32 No. 2, pp. 112-118. https://doi.org/10.1108/01445151211212262

[9] Geoffrey Boothroyd, Peter Dewhurst, and Winston A. Knight. 2010. Product Design for Manufacture and Assembly (3rd ed.). CRC Press. https://doi.org/10.1201/9781420089288

[10] Ruei-Che Chang, Chih-An Tsao, Fang-Ying Liao, Seraphina Yong, Tom Yeh, and Bing-Yu Chen. 2021. Daedalus in the Dark: Designing for Non-Visual Accessible Construction of Laser-Cut Architecture. In The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21). Association for Computing Machinery, New York, NY, USA, 344–358. DOI:https://doi.org/10.1145/3472749.3474754

[11] Weikai Chen, Yuexin Ma, Sylvain Lefebvre, Shiqing Xin, Jonàs Martínez, and wenping wang. 2017. Fabricable tile decors. ACM Trans. Graph. 36, 6, Article 175 (November 2017), 15 pages. DOI:https://doi.org/10.1145/3130800.3130817

[12] Paolo Cignoni, Nico Pietroni, Luigi Malomo, and Roberto Scopigno. 2014. Field-aligned mesh joinery. ACM Trans. Graph. 33, 1, Article 11 (January 2014), 12 pages. DOI:https://doi.org/10.1145/2537852

[13] Chi-Wing Fu, Peng Song, Xiaoqi Yan, Lee Wei Yang, Pradeep Kumar Jayaraman, and Daniel Cohen-Or. 2015. Computational interlocking furniture assembly. ACM Trans. Graph. 34, 4, Article 91 (August 2015), 11 pages. DOI:https://doi.org/10.1145/2766892

[14] Ruslan Guseinov, Eder Miguel, and Bernd Bickel. 2017. CurveUps: shaping objects from flat plates with tension-actuated curvature. ACM Trans. Graph. 36, 4, Article 64 (July 2017), 12 pages. DOI:https://doi.org/10.1145/3072959.3073709

[15] P.N. Hasluck. The Handyman's Guide: Essential Woodworking Tools and Techniques. Skyhorse Publishing, 2011. isbn: 9781626366589. url: https://books.google.de/books?id=oR0nAgAAQBAJ.

[16] Hildebrand, K., Bickel, B. and Alexa, M. (2012), crdbrd: Shape Fabrication by Sliding Planar Slices. Computer Graphics Forum, 31: 583-592. https://doi.org/10.1111/j.1467-8659.2012.03037.x

[17] Robert Kovacs, Anna Seufert, Ludwig Wall, Hsiang-Ting Chen, Florian Meinel, Willi Müller, Sijing You, Maximilian Brehm, Jonathan Striebel, Yannis Kommana, Alexander Popiak, Thomas Bläsius, and Patrick Baudisch. 2017. TrussFab: Fabricating Sturdy Large-Scale Structures on Desktop 3D Printers. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17). Association for Computing Machinery, New York, NY, USA, 2606–2616. DOI:https://doi.org/10.1145/3025453.3026016

[18] Weisheng Lu, Tan Tan, Jinying Xu, Jing Wang, Ke Chen, Shang Gao & Fan Xue (2021) Design for manufacture and assembly (DfMA) in construction: the old and the new, Architectural Engineering and Design Management, 17:1-2, 77-91, DOI: 10.1080/17452007.2020.1768505

[19] Shiran Magrisso, Moran Mizrahi, and Amit Zoran. 2018. Digital Joinery For Hybrid Carpentry. Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, Paper 167, 1–11. DOI:https://doi.org/10.1145/3173574.3173741

[20] James McCrae, Nobuyuki Umetani, and Karan Singh. 2014. FlatFitFab: interactive modeling with planar sections. In Proceedings of the 27th annual ACM symposium on User interface software and technology (UIST '14). Association for Computing Machinery, New York, NY, USA, 13–22. DOI:https://doi.org/10.1145/2642918.2647388

[21] Stefanie Mueller, Bastian Kruck, and Patrick Baudisch. 2013. LaserOrigami: laser-cutting 3D objects. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13). Association for Computing Machinery, New York, NY, USA, 2585–2592. DOI:https://doi.org/10.1145/2470654.2481358

[22] Wayne Niblack and John Yin. A pseudo-distance measure for 2D shapes based on turning angle. Proceedings., International Conference on Image Processing, 1995, pp. 352-355 vol.3, doi: 10.1109/ICIP.1995.537646.

[23] Martin Nisser, Christina Chen Liao, Yuchen Chai, Aradhana Adhikari, Steve Hodges, and Stefanie Mueller. 2021. LaserFactory: A Laser Cutter-based Electromechanical Assembly and Fabrication Platform to Make Functional Devices & Robots. Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, Article 663, 1–15. DOI:https://doi.org/10.1145/3411764.3445692

[24] Davy D. Parmentier, Bram B. Van Acker, Jan Detand, and Jelle Saldien. Design for assembly meaning: a framework for designers to design products that support operator cognition during the assembly process. Cogn Tech Work 22, 615–632 (2020). https://doi.org/10.1007/s10111-019-00588-x

[25] Jesús Pérez, Miguel A. Otaduy, and Bernhard Thomaszewski. 2017. Computational design and automated fabrication of kirchhoff-plateau surfaces. ACM Trans. Graph. 36, 4, Article 62 (July 2017), 12 pages. DOI:https://doi.org/10.1145/3072959.3073695

[26] Thijs Roumen, Ingo Apel, Jotaro Shigeyama, Abdullah Muhammad, and Patrick Baudisch. 2020. Kerf-Canceling Mechanisms: Making Laser-Cut Mechanisms Operate across Different Laser Cutters. In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST '20). Association for Computing Machinery, New York, NY, USA, 293–303. DOI:https://doi.org/10.1145/3379337.3415895

[27] Thijs Roumen, Jotaro Shigeyama, Julius Cosmo Romeo Rudolph, Felix Grzelka, and Patrick Baudisch. 2019. SpringFit: Joints and Mounts that Fabricate on Any Laser Cutter. In Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19). Association for Computing Machinery, New York, NY, USA, 727–738. DOI:https://doi.org/10.1145/3332165.3347930

[28] Rundong Tian, Sarah Sterman, Ethan Chiou, Jeremy Warner, and Eric Paulos. 2018. MatchSticks: Woodworking through Improvisational Digital Fabrication. Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, Paper 149, 1–12. DOI:https://doi.org/10.1145/3173574.3173723

[29] Remco C. Veltkamp. Shape matching: similarity measures and algorithms. Proceedings International Conference on Shape Modeling and Applications, 2001, pp. 188-197, doi: 10.1109/SMA.2001.923389.

[30] Wang, Z., Song, P. and Pauly, M. (2021), State of the Art on Computational Design of Assemblies with Rigid Parts. Computer Graphics Forum, 40: 633-657. https://doi.org/10.1111/cgf.142660

[31] Wei Yan. Augmented Reality Applied to LEGO Construction: AR-based Building Instructions with High Accuracy & Precision and Realistic Object-Hand Occlusions. arXiv preprint arXiv:1907.12549 (2019).

[32] Emily Whiting, John Ochsendorf, and Frédo Durand. 2009. Procedural modeling of structurally-sound masonry buildings. In ACM SIGGRAPH Asia 2009 papers (SIGGRAPH Asia '09). Association for Computing Machinery, New York, NY, USA, Article 112, 1–9. DOI:https://doi.org/10.1145/1661412.1618458

[33] Randall H. Wilson and Toshihiro Matsui. Partitioning An Assembly For Infinitesimal Motions In Translation And Rotation, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 1992, pp. 1311-1318, doi: 10.1109/IROS.1992.594555.

[34] Jiaxian Yao, Danny M. Kaufman, Yotam Gingold, and Maneesh Agrawala. 2017. Interactive Design and Stability Analysis of Decorative Joinery for Furniture. ACM Trans. Graph. 36, 2, Article 20 (April 2017), 16 pages. DOI:https://doi.org/10.1145/3054740

[35] Clement Zheng, Ellen Yi-Luen Do, and Jim Budd. 2017. Joinery: Parametric Joint Generation for Laser Cut Assemblies. In Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition (C&C '17). Association for Computing Machinery, New York, NY, USA, 63–74. DOI:https://doi.org/10.1145/3059454.3059459