Interlocking Structure Assembly with Voxels

Yinan Zhang[1] and Devin Balkcom[2]

***Abstract*— This paper explores the problem of building a structure of a desired shape, using re-usable interlocking blocks. Blocks are cubes; we make use of nine different types of cubes, each with different arrangements of male and female connectors on the six sides of the cube. The desired shape is specified by a set of voxels. We propose an algorithm that lays out cubes in a particular pattern to give the desired shape and gives a fairly simple assembly order.**

I. Introduction

Furniture, buildings, and machines are typically built out of smaller pieces, allowing complex structures to be built from a small selection of easily manufacture-able parts. These smaller parts form a language for the design of devices; the range of what can be built depends on what parts are available. Some parts, such as screws and nuts, are intended to allow easy dis-assembly as well as assembly, to allow repair, replacement, or re-use. Parts may be connected in different ways: by glue, cement, or geometric constraints.

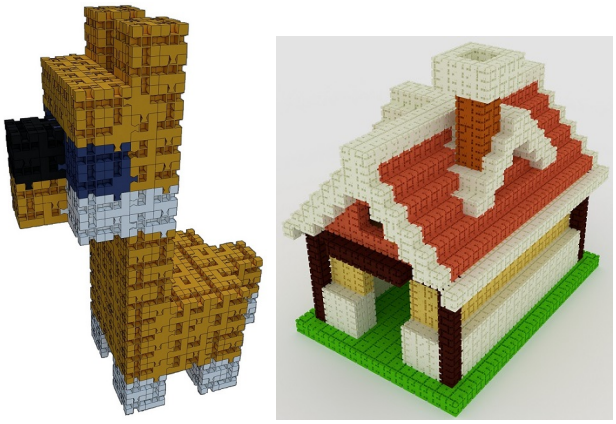
In this work, we examine the problem of building structures of arbitrary geometric shape (specified with voxels) using cube-shaped pieces that include connectors. The structures built are connected using interlocking geometries rather than glue, cement, or friction, to allow easy assembly and dis-assembly. This work is motivated by recent increase of availability of 3D printing technology: 3D printers allow relatively fast printing of small objects that fit within the working volume of the printer; we would like to build larger structures quickly, with the possibility of re-use of parts after dis-assembly.

Systems like Legos are designed to allow physical experimentation, assembly, and prototyping at larger scales. However, these systems often rely on frictional locking that can be hard to achieve, and unreliable for large systems when external forces are applied. We therefore explore structures that interlock and are constrained to the desired shape geometrically. We imagine building structures of the type described in this paper using robots (see Figure 11), although the current work focuses on the design process.

The input for our problem is a set of voxels that is a discretized representation of an object of interest. The desired output is a set of pieces, piece types, locations and an assembly order such that the pieces cover, and only cover the voxels, with the possible exception of some connectors.

We will use eight blocks to build each voxel; blocks are unit cubes, and voxels are $2 \times 2 \times 2$ cubes. Our goal is to construct an assignment of block types for each piece such that these blocks can be assembled one by one to mimic the structure of any 3D model and no sub-structure of the resulting assembly can be disassembled without removing some *key* blocks. Figure 1 shows two interlocking structures generated by the design algorithms based on simple voxelized models of a quadruped animal (an alpaca) and a house.

[1,2]Department of Computer Science, Dartmouth College, NH, USA



(a) An alpaca model with 400 blocks, with one key at the top of each ear.

(b) A model of a house with 13104 blocks, with one key at the top of the chimney.

Fig. 1: Two globally interlocking models.

Each block has six faces and may be adjacent to at most six blocks. For any pair of adjacent blocks, we want to assign a *joint* (made of two compatible connectors) that constrains the relative motion of blocks. For a block with joint assignments on its faces, there is a corresponding *block type* that can be built and reused.

The completed structure will have a small number of *keys*: pieces that, if they all remain fixed relative to any non-key piece, ensure that the entire structure behaves as a single rigid body. A structure is said to be *locked* if all keys are immobilized.

The basic unit of our design is four interlocking blocks, called a *square*. By appending squares, we can build an interlocking *segment*. The algorithm constructs the model segment by segment to build interlocking layers that comprise the model. The run-time of our algorithm is linear in the number of voxels.

II. Related Work

Friction connectors [11], [10], glue, and 3D interlocking shapes [16] are three methods commonly proposed to connect 3D printed parts.

This work is closest in spirit to that of Song *et al.* [16], [17], which gives an algorithm for constructing interlocking

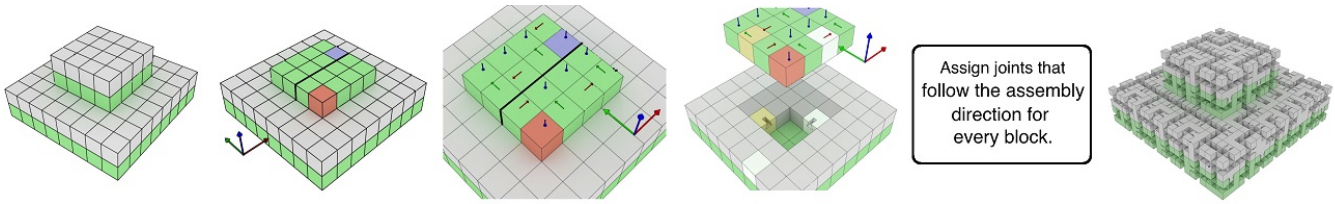


Fig. 2: We firstly work on even layers, determine segments and assembly direction of blocks. Then ensure the even layers lock with lower-layers by potentially removing some squares. Constraints on segments and block assembly directions in odd layers are then determined. Finally, the whole structure is constructed using prefabricated blocks that satisfy the constraints.

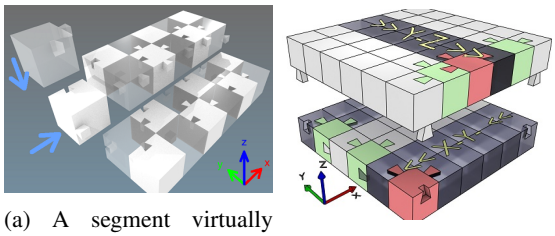
3D geometric puzzles. Both approaches use keys (pieces that must be immobilized using external means) to lock sets of pieces geometrically, and both allow shapes represented using voxels to be built. However, in [16], the shapes of the pieces are the primary output of the algorithm, meaning that pieces must be constructed individually; in our work, there are only a few pre-defined types of pieces, allowing piece re-use.

This work is also inspired by the study of self-assembly robots. Popescu *et at.* [12] work on digital materials which explored a design of 2D components to construct 3D structures. Rus and Vona's early work on the Crystalline robot [14] presented an algorithm to assemble robotic module. Recently Rubenstein et al. [13] demonstrated an algorithm of moving kilo-bots to form certain planar shapes. Arbuckle et al. [1] allowed identical memoryless agents to construct and repair arbitrary shapes in the plane.

Interlocking puzzles have a rich history. Interlocking burr puzzles, consisting of notched sticks, have been used in toys, furniture and architecture for centuries. In China and Japan, wood joints are commonly seen in traditional architecture, as well as furniture [19]. Timber connections have been claimed to improve seismic characteristics [6]. Recently, architects such as Kengo Kuma [8] have designed large-scale buildings inspired by traditional puzzles.

Xin *et al.* [18] suggest that Culter [2] was the first to employ computers to systematically analyze all possible combinations of the 6-piece burr puzzle. Work by Xin *et al.* [18] generalized the 6-piece orthogonal burr puzzle to design and model burr puzzles from 3D models.

Computer scientists have been interested in puzzles for many years [3], [5], [7]. Early works primarily focus on the solution of puzzles; study of design algorithms is relatively recent. Lau *et al.* [9] proposed an algorithm that takes as input a 3D model of a man-made object, and automatically generates parts and connectors. Luo *et al.* [11] also partitions an object into parts, but further requires the dimensions of each part fit the working volume of a 3D printer. Saul *et al.* [15] provided a sketch-based interface and design-validation tools to help novice user design their own chairs and fabricate them from sheet materials. Fu *et al.* [4] proposed a method to plan and construct a network of joints in the design of interlocking furniture. By repeatedly constructing locally interlocking groups, the structure can be globally interlocked. However, the formulation is NP-hard in the number of pieces.



(a) A segment virtually lifted. The higher segment constrains the movement of the lower one in Z+ direction and XY plane.

(b) A layer virtually lifted. Red pieces are keys of each component, green ones are segment keys.

Fig. 3: Layers and segments. In the same layer component, later constructed segment constrains previous segments. Between layers, male joints immobilize both layers.

III. Overview

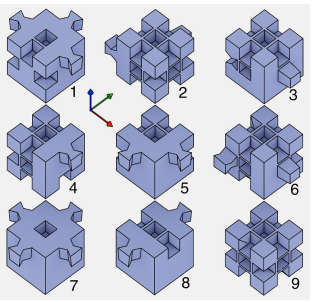
Figure 2 gives a visual description of the overall process. The goal of our work is to assign predefined blocks with male and female connectors to build an interlocking structure and to generate the order of the assembly. Assembly is accomplished using a simple coordinate-aligned translation of each block.

A *joint* is a pair of male-female connectors on two facets of blocks. Each joint permits motion in a direction either normal to the face, or along one of the four coordinate directions tangent to the face (Figure 5).

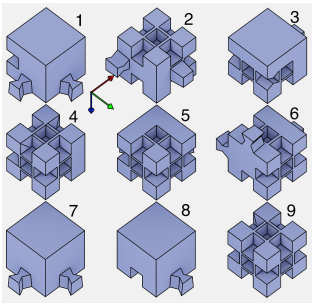
Each *layer* is a set of horizontal blocks. Inside each layer, we would like to build a structure such that, if the first and last blocks are immobilized with respect to their relative position, the whole layer is immobilized. We built layers out of horizontal strips, called *segments*. We assign the last block of each segment based on the order of segments. Every segment's movement is prevented by a following adjacent segment.

We assemble blocks one vertical layer at a time. Every layer has male tangential connectors that lock with its substrate (the layer below) to be immobilized.

Because tangential joints between layers require sliding movements in the horizontal plane, blocks with male connectors in the upper layer are only possible if their lower layer neighbors have no adjacent block in the sliding direction. In every pair of layers, the odd layer always has a substrate with the same shape, since each voxel layer is divided into a pair



(a) X+, Y- and Z+ faces of the 9 kinds of blocks



(b) X-, Y+ and Z- faces of the 9 kinds of blocks

Fig. 4: The nine block types.

of block layers, allowing tangential joints can be assigned. But even layers and their substrates do not have such a guarantee. To fix this, we need to firstly know the assembly directions of every block in the even layer. Then we delete some squares in the substrate accordingly to allow the upper layer blocks to slide into place.

We will also describe a way of assigning joints between blocks based on segment order and last block assembled. Assembly directions and orders of blocks are implied by the joint assignments.

Predefined blocks with connectors are then selected to be consistent with the joint assignments between blocks. And we assemble the whole model with these predefined blocks.

This process is described in Section VIII-A in pseudo-code.

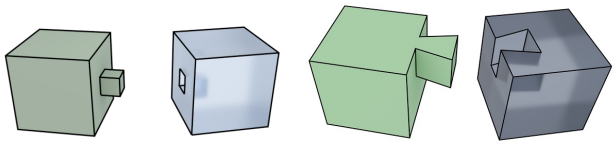
IV. Blocks

In this and the next several sections, we will show how to construct a simple cube-shaped model out of blocks arranged into squares, segments, and ultimately, layers. Then, in Section VIII, we will present an algorithm that breaks a particular model into layers and assigns block types to build those layers in sequence.

Figure 4 shows the nine types of blocks that we use to construct models. Various pairs of blocks fit together along certain facets, depending on the available joints on each facet.

A. Joints

Blocks can be described by the joint types available along each facet. Each joint between a pair of blocks permits motion in a direction either normal to the face, or along one of the four coordinate directions tangent to the face. There are therefore four types of connectors that make up joints: male and female connectors that permit motion in the normal direction, and male and female connectors that permit a single tangential motion. Figure 5 shows tangential and normal connectors. Each block type is defined by the arrangement of connectors on faces. We denote a joint between two blocks a and b as $J(a,b)$. In Fig. 5a, $J(a,b) = X_-$ meaning that a can be dis-assembled in $X-$ direction and b can be dis-assembled in $X+$ direction. $J(a,b) = Z_+$ in Fig. 5b. $J(b,a) = -J(a,b) = Z_-$.



(a) A normal direction joint (b) A tangential direction joint.

Fig. 5: Two kinds of connectors: one permits motions along the normal direction, while the other permits motion in tangential direction along a coordinate.

V. Squares

A square is made up of four blocks arranged in a 2x2 pattern within a layer. Depending on the selection and orientation of blocks of various types, the square might be assembled using only vertical translation, or may instead require that blocks be added from the side. Table I lists joints of 4 different squares, and their assembly orders. Assembly directions are implicitly determined by the order and joints.

In order to determine which block to use in a square, we need to fully understand how a square is connected to all of its six neighbor squares. We assign joints between blocks before deciding the types of blocks. After all the joints are decided, we finally match each block with the predefined block types.

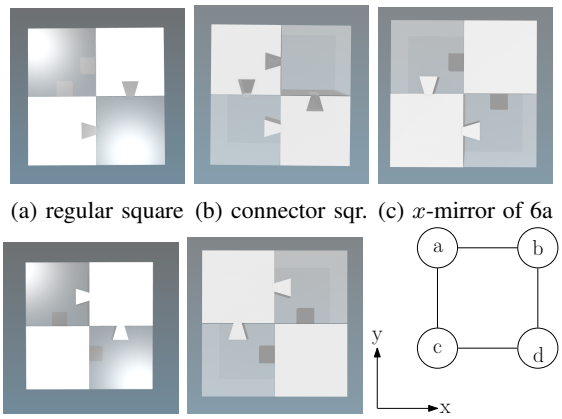
Based on the internal joints, we classify squares into two types: *regular squares*, and *connector squares*. There are four regular squares, shown in Figure 6; each has one x-axis joint, one y-axis joint, and two z-axis joints (both on the same block). There are four connector squares, each containing four z-axis joints. Figure 6b shows one example of a connector square; other connector squares are symmetric to this one.

Regular squares require some clearance in the x and y directions for assembly. For example, Fig. 6a requires the $X+$ and $Y-$ direction to have no adjacent square built when assembling the square. A connector square does not have such a requirement because all blocks are translated along the z-axis.

Keys of regular squares are the last blocks assembled. Connector squares do not have keys, but the last block still immobilizes the square if no block can move in the $Z-$ direction and the last block is prevented from moving in the $Z+$ direction. These conditions must be enforced by blocks external to the square.

A *pseudo-key* is a block which, if immobilized with respect to any single block in the lower layer, prevents any block in the assembly from moving in the $Z+$ direction. The last block of a connector square is a pseudo-key. Connector squares have male connectors that by nature prevent $Z-$ movement of blocks.

Notationally, a square s has four pieces $s(a), s(b), s(c)$ and $s(d)$, ordered as shown in Fig. 6f. There are some symmetries between squares; Fig.6c is a mirror of Fig. 6a in the x direction. Table I lists joints of four symmetric regular square. Similarly, we can mirror connector squares.



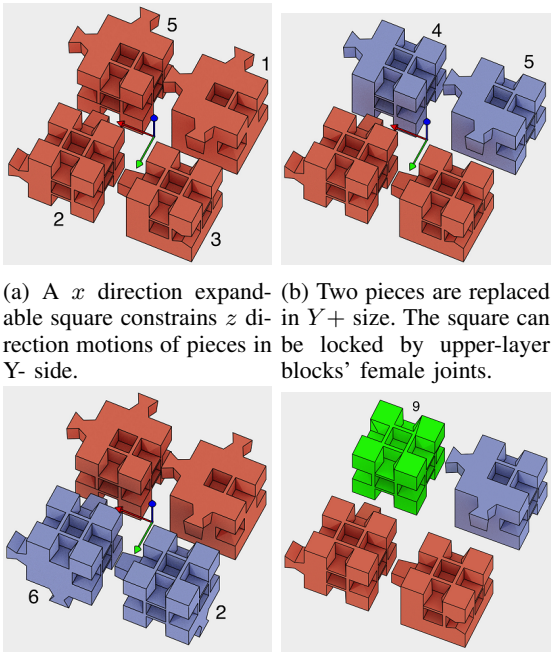
(a) regular square (b) connector sqr. (c) x-mirror of 6a

(d) y-mirror of 6a (e) x-mirror of 6d (f) Notation

Fig. 6: Two types of squares and their mirrors. Fig. 6c - 6e are mirrors of Fig. 6a.

Square	Fig 6a	Fig 6c	Fig 6d	Fig 6b
$J(a,b)$	$X-$	$X-$	$Z-$	$Z-$
$J(a,c)$	$Y+$	$Z-$	$Y+$	$Z-$
$J(c,d)$	$Z-$	$Z+$	$X-$	$Z-$
$J(b,d)$	$Z-$	$Y+$	$Z+$	$Z+$
Order	(c,a,b,d), (a,c,b,d)	(d,b,a,c), (b,d,a,c)	(a,c,d,b), (c,a,d,b)	(a,c,d,b)
Key	d	c	b	b (pseudo-key)

TABLE I: Joints of four squares



(a) A x direction expandable square constrains z direction motions of pieces in Y- side.

(b) Two pieces are replaced in $Y+$ size. The square can be locked by upper-layer blocks' female joints.

(c) Lock lower layer pieces by replacing the other two pieces with male joints on Z- sides.

(d) A square that can be locked by any pieces to prevent z-directional movements.

Fig. 7: Regular squares built using 7 kinds of pieces.

A. Block types

The ultimate goal is to connect sequences of squares to form segments and layers that are themselves components of the desired model. After joints are assigned, blocks must be selected to match the joint assignment; this motivates the design of the nine block types shown in Figure 4.

We use regular squares to: (1) connect a list of regular squares to form a segment with one key, (2) constrain the $Z+$ movement of adjacent squares that are not in the same segment, (3) lock with lower layer adjacent squares to constrain movement in Z axis, and (4) be locked by upper layer adjacent blocks.

Fig. 7a is an example of a regular square with male connectors in the $X+$ and $Y-$ directions. The $X+$ connectors link a with regular neighbor square in $X+$ direction and immobilize two adjacent blocks. The $Y-$ connectors prevent adjacent square in $Y-$ direction from moving in the $Z+$ direction. Replacing the top two blocks in Fig. 7a gives a square with 3-side female connectors as shown in Fig. 7b. This square is used when there is not adjacent square in Y-direction, so it can be locked by male connectors from the upper layer. Fig. 7c shows two male connectors that lock with lower layers. Fig. 7d is an example of introducing female connectors in all four edges.

Connector squares are used to link one or more squares and prevent their movements in the $Z+$ direction and in the XY plane.

We assign male connectors to some or all of the 2x1 facets of connector squares (the faces pointing outwards in the layer). On any face, we place either 0 or 2 male connectors. There are five situations: connectors on four sides, connectors on three sides, connectors on two sides (two sub-cases) and connectors on a single side. The final case can be eliminated by placing some additional connectors on sides that are adjacent to empty locations. Figure 8 shows the four kinds of connector squares.

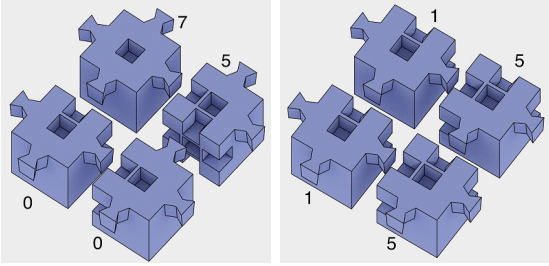
VI. Segments

We build long flat chains of squares (named *segment*) by connecting joints of previous squares to new squares.

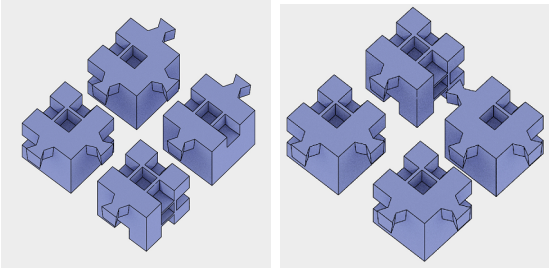
A *maximal segment* is a segment not contained by other segments. An *x-segment* is a maximal segment parallel to the x-axis, and a *y-segment* is parallel to the y-axis. Even layers have only x-segments while odd layers have only y-segments.

Ultimately, segments will be used to construct flat layers of blocks. Sometimes segments need to be built between two adjacent pre-existing segments. Since regular squares require clearance in the horizontal direction for assembly, we build such intermediate *connector segments* purely out of connector squares, which permit assembly in the purely vertical direction. As we will see in the discussion of the assembly algorithm, there are other circumstances in which there is insufficient clearance to permit horizontal assembly, and connector segments are needed.

We also need segments with regular squares. Regular squares contain blocks sliding in horizontal plane. Adding male connectors to these sliding blocks can constrain the



(a) Male joints on 4 sides (b) Male joints on 3 sides



(c) Male joints on 2 sides (d) Male joints on 2 sides

Fig. 8: Connectors squares built using 6 kinds of pieces.

movement of current segment/layer in z-axis. Other segments are *regular segments*.

In section VII, we describe in detail how to determine the construction order of segments in the same layer, as well as the choice of segment key (or pseudo-key) based on the order.

We choose a local coordinate frame for each layer such that segments are aligned with the x axis. A regular segment with the key at the $X+$ end and $Y-$ side of the segment is shown in Figure 9a. We build this segment along the x-positive direction, one square at a time, and denote the segment as X_+Y_- based on the key location. Similarly, we have X_+Y_+, X_-Y_+ and X_-Y_- segments. For some of our segment designs, the pseudo-key is not at either end of the segment. Such segments are labelled as $X_\pm Y_-$ or $X_\pm Y_+$ (Fig. 9b); a connector square contains the pseudo-key.

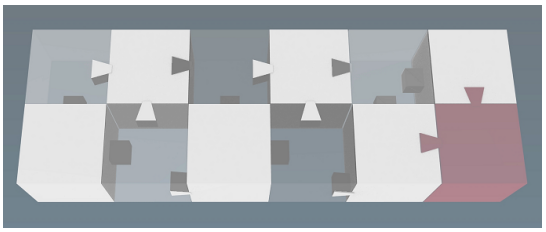
A connector x-segment can be X_+Z, X_-Z or $X_\pm Z$, depending on the pseudo-key position. Fig. 9c is an example of an X_+Z segment with a pseudo-key at the $X+$ end.

A regular segment has two pieces that can slide in the XY plane; these two pieces can have male connectors to prevent motion in the Z direction. Knowing the type of a regular segment means knowing where to place male connectors to lock with lower layer. For example, a X_+Y_- segment with n squares ($n \geq 2$) has $2n-2$ prevents (at the $Y-$ side but not at either ends) sliding in from $Y-$ direction.

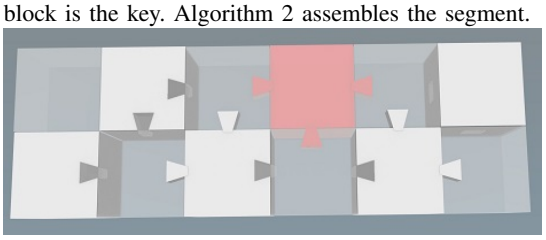
VII. Layers

A layer is a set of blocks with the same z-coordinate. A layer can have multiple components. We build even layers with only x-segments and odd layers with y-segments.

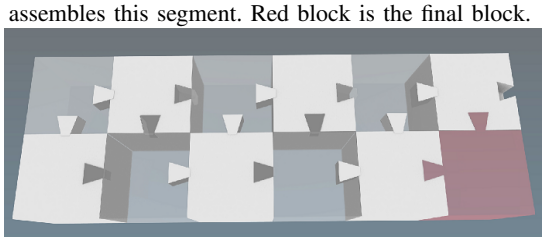
We construct a rigid layer by connecting a set of segments in the same plane; each segment prevents movement of the previous segment. To determine the assembly order inside a



(a) An X_+Y_- segment with all regular squares. Red block is the key. Algorithm 2 assembles the segment.



(b) An $X_\pm Y_+$ segment $l = [s_0, s_1, s_2]$ can be viewed as a X_+Y_- segment $l' = [s_0]$ and a X_-Y_- segment $l'' = [s_2]$ connected by a connector square s_1. Alg. 3 assembles this segment. Red block is the final block.



(c) An X_+Z segment with all connector squares. Alg. 4 assembles this segment. Red block is the final block.

Fig. 9: Segment types explained.

layer by selecting the last block assembled then determining the dis-assembly order of segments. The final block will be constrained by the following layer.

For even-layer components, the last block will always be a block at either end of a *boundary segment* with the largest y-coordinate, where a *boundary segment* is a maximal segment with same-layer neighbor segment(s) on exactly one side. In an odd layer component, if one end of the boundary segment with the largest x-coordinate block has a neighbor in the layer above, we select that block as the final block. Otherwise, we choose any other block with an upper layer neighbor. Some odd components have no upper layer neighbors, so we choose the block at one end of a boundary segment with the largest x-coordinate.

After deciding the last assembled block in the layer, we determine the order of segment construction, and then assign joints to prevent the $Z+$ movement of each segment using its successor's male connectors. Fig. 10 has two layers, each with three segments. The male connectors prevents movement in XY plane.

A. Segment order

We find the segment order by determining a dis-assembly order. The segment containing the component's last block is the last one assembled. Any unscheduled neighbors of a scheduled segment are assembled before it.

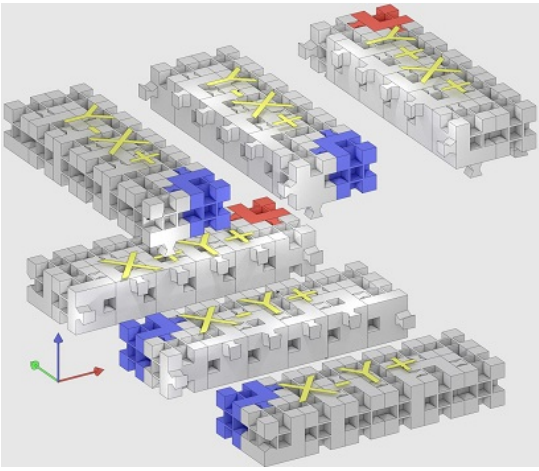


Fig. 10: A two-layer structure. Layers and segments are virtually separated with male joints highlighted. The text shows the type of each segment. Blue pieces are segment keys and red pieces are component keys.

Let s be a square in a segment l adjacent to a later assembled segment. The last assembled block of l can be either block in s adjacent to the next segment's blocks assembled from $Z+$ direction. We choose the block that generates fewer connector squares, which makes the next step easier.

B. Model fix

Segment type can be determined when the key (or pesudo-key) of a segment is known along with the construction order of its neighbors. Section VIII-C assigns joints between blocks inside a segment based on the segment type. Therefore the block assembly directions are implicitly decided after knowing the segment type. Let $l = [s_0, s_1, \ldots, s_n]$ be a X_+Y_- segment, where $n \in \mathbb{Z}_+$. These blocks are assembled from the $X+$ direction:

- $\{s_i(d) | i \in \{0..n-1\}\}$
- $s_n(b)$

These blocks are assembled from $Y+$ direction: $\{s_i(a) | i \in \{0..n\}\}$. Blocks assembled within the XY plane are *sliding blocks*. Other blocks are assembled from Z direction.

In order to lock with lower layer, we need to ensure that in the assembly direction, the sliding blocks' lower neighbors do not have neighbors in two different horizontal directions.

We only need to perform the check for even layer components, because odd layers have lower layers with the same shape. If this condition cannot be satisfied, we firstly expand the layer and its upper layer to at least 2×2 square size, or find a 2×2 square set with no connector squares. Then we remove the square in the lower layer according to the assembly direction of blocks to ensure there can be tangential connectors with two different sliding directions.

C. Joints between segments and layers

A x-segment with the last block in the $Y+$ side requires the $Y+$ side neighbors to be built after current segment. Similarly, the $Y-$ side last block requires $Y-$ clearance. If such condition is not satisfied, the current segment must be a connector segment.

Inside a layer, to ensure that a segment cannot move in $Z+$ direction, we assign joints between the segment and its later-built neighbor segment. In Figure 10, some segments have male joint connector neighbor segments; if the segment is immobilized in Z axis, its neighbor cannot move in $Z+$ direction.

The last step of working on a layer is to assign tangential joints between the current layer and the layer below. For any sliding block with assembly direction d, if its neighbor in the lower layer does not have a neighbor in $-d$ direction, we can assign a tangential joint between the block and its lower layer neighbor. Figure 10 shows some tangential joints between layers.

Inside each layer, some subset of segments can potentially move in $Z+$ direction. By connecting both ends of a upper layer segment with the lower layer, no segments can move along z-axis.

VIII. Automatic design algorithm

A. Algorithm overview

Algorithm 1 gives an overview of the design algorithm.

Algorithm 1 Algorithm overview

```
function ConstructInterlockingAssembly(M)
    M' ← split every voxel into 8 equal size blocks.
    for each layer L_i from top to bottom do
        Determine the last block assembled of the layer.
        Order segments in the layer
        Determine the last block of each segment.
        Determine assembly directions of each block.
        Test if there can be joints between L and L_{i-1}
        if test failed then
            remove some squares in L_i according to block assembly directions in L.
        Assign joints to blocks in every segment
        Assign joints between segments
        Assign joints between layers.
    Match every block with predefined blocks.
    Assemble the model using predefined blocks.
```

B. Symmetries

Some structures are similar to each other. The similarity helps simplify the algorithm for assigning joints between blocks.

An *x-mirror* of a structure is the reflection formed by placing a plane mirror perpendicular to the x-axis, while a *y-mirror* is the reflection by a mirror perpendicular to the y-axis. For squares, Figure 6d is a y-mirror of Figure 6a, Figure 6e is an x-mirror of Figure 6d.

For segments, X_+Y_- and X_-Y_- are x-mirrors. X_+Y_+ and X_-Y_+ are x-mirrors. X_+Y_- and X_-Y_+ are y-mirrors, as are X_-Y_- and X_-Y_+. So for these four kinds of

segments, we need an algorithm to assign joints for only one type. $X_{\pm}Y_{+}$ and $X_{\pm}Y_{-}$ segments are y-mirrors to each other, so we can assign joints for $X_{\pm}Y_{-}$ by assigning a mirror $X_{\pm}Y_{+}$ segment. Similarly, $X_{+}Z$ and $X_{-}Z$ segments are x-mirrors.

C. Joints in a segment and between layers

We choose a local coordinate frame for each layer such that segments are aligned with the x-axis, based on the segment order and last block. We have six types of regular segments: $X_{+}Y_{+}$, $X_{+}Y_{-}$, $X_{-}Y_{+}$, $X_{-}Y_{-}$, $X_{\pm}Y_{+}$ and $X_{\pm}Y_{-}$, and 3 types of connector segments: $X_{+}Z$, $X_{-}Z$ and $X_{\pm}Z$.

Algorithm 2 assigns joints between blocks in a $X_{+}Y_{-}$ segment. The result is shown in Figure 9a. As mentioned in Section VIII-B, we need a algorithm for only one type of segment to handle all segment types.

A $X_{\pm}Y+$ segment can be seen as a $X_{+}Y_{+}$ segment and a $X_{-}Y_{+}$ segment connected by a connector square. Algorithm 3 describes joints in a $X_{\pm}Y_{-}$ segment. Similarly, a $X_{\pm}Y_{+}$ segment can be viewed as a y-mirror of a $X_{\pm}Y_{-}$ segment.

Assigning joints to connector segments is much simpler, because all are z-axis joints. Algorithm 4 assigns joints to an $X_{+}Z$ segment. $X_{-}Z$ segment is an x-mirror of an $X_{+}Z$ segment.

Algorithm 2 Algorithm to assemble a $X_{+}Y_{-}$ segment

function ASSEMBLEXPYN(l)
  for $i \in [0, n-1]$ **do**
    // Assign joints inside each square
    $J(s_i(c), s_i(a))$= Y_{-}
    $J(s_i(c), s_i(d))$= X_{-}
    $J(s_i(b), s_i(a))$= $J(s_i(b), s_i(d))$ = Z_{+}
    // Assign joints to lock previous square
    if $i \geq 1$ **then**
      $J(s_i(a), s_(i-1)(b))$ = Z_{+}
      $J(s_i(c), s_(i-1)(d))$ = Y_{-}
    // Lock the square in neighbor segment
    if $n(s_i(a), Y+)$ exists & assembled **then**
      $n_a \leftarrow n(s_i(a), Y+)$
      $n_b \leftarrow n(s_i(b), Y+)$
      $J(s_i(a), n_a)$ = $J(s_i(b), n_b)$ = Z_{+}
  // Lock the square containing the key
  $J(s_n(c), s_n(a))$= Y_{-}
  $J(s_n(a), s_n(b))$= X_{-}
  $J(s_n(d), s_n(c))$= $J(s_i(d), s_i(b))$ = Z_{+}
  $J(s_n(a), s_(n-1)(b))$ = Z_{+}
  $J(s_n(c), s_(n-1)(d))$ = Y_{-}
  if $n(s_n(a), Y+)$ exists & assembled **then**
    $n_a \leftarrow n(s_n(a), Y+)$
    $n_b \leftarrow n(s_n(b), Y+)$
    $J(s_n(a), n_a)$ = Z_{+}
    if not $n(n_b, X+)$ exist **then**
      $J(s_n(b), n_b)$ = X_{+}

Algorithm 3 Algorithm to assemble a $X_{\pm}Y_{+}$ segment

function ASSEMBLEXPNYN(l, s_k)
  $l' \leftarrow [s_0, \ldots, s_{k-1}]$
  $l'' \leftarrow [s_{k+1}, \ldots, s_n]$
  Assemble l' as a $X_{+}Y_{-}$ segment.
  Assemble l'' as a $X_{-}Y_{-}$ segment.
  Assign a Type II square or Type II square mirror to s_k with the same key position.
  $J(s_k(a), s_{k-1}(b)) = Z_{+}$
  $J(s_k(c), s_{k-1}(d)) = Z_{+}$
  $J(s_k(b), s_{k+1}(a)) = Z_{+}$
  $J(s_k(d), s_{k+1}(c)) = Z_{+}$
  if $n(s_k(a), Y+)$ exists & assembled **then**
    $J(s_k(a), n(s_k(a), Y+)) = Z_{+}$
  if $n(s_k(c), Y-)$ exists & assembled **then**
    $J(s_k(c), n(s_k(c), Y-)) = Z_{+}$
  if $n(s_k(b), Y+)$ exists & assembled **then**
    $J(s_k(b), n(s_k(b), Y+)) = Z_{+}$
  if $n(s_k(d), Y-)$ exists & assembled **then**
    $J(s_k(d), n(s_k(d), Y-)) = Z_{+}$

Algorithm 4 Algorithm to assemble a $X_{+}Z$ segment

function ASSEMBLEXPZ(l)
  for $i \in [0, n]$ **do**
    if $i > 0$ **then**
      // lock previous square
      $J(s_i(a), s_{i-1}(b)) = Z_{+}$
      $J(s_i(c), s_{i-1}(d)) = Z_{+}$
    $J(s_i(b), s_i(a)) = Z_{+}$
    $J(s_i(c), s_i(a)) = Z_{+}$
    $J(s_i(d), s_i(b)) = J(s_i(d), s_i(c)) = Z_{+}$
    // Lock adjacent segments
    if $n(s_i(a), Y+)$ exist & assembled **then**
      $J(s_i(a), n(s_i(a), Y+)) = Z_{+}$
      $J(s_i(b), n(s_i(b), Y+)) = Z_{+}$
    if $n(s_i(c), Y-)$ exist & assembled **then**
      $J(s_i(c), n(s_i(c), Y-)) = Z_{+}$
      $J(s_i(d), n(s_i(d), Y-)) = Z_{+}$
  Re-assign Z directional joints in s_n pieces to have the pre-determined key piece position.

Let l and l' be two adjacent segments ordered such that l is assembled before l'. Let Z be the set of blocks in l' that are assembled from $Z+$ direction. To connect l with l', we simply assign Z directional joints to any block in Z and its adjacent block in l.

Let p be a sliding block in the current layer with disassembly direction d whose lower layer neighbor p' does not have a neighbor in the d direction. We assign a tangential joint between p and p' to lock the two layers.

IX. RESULTS

We implemented the design algorithm in Python and executed it on a desktop with a 3.4GHz CPU and 8GB memory.

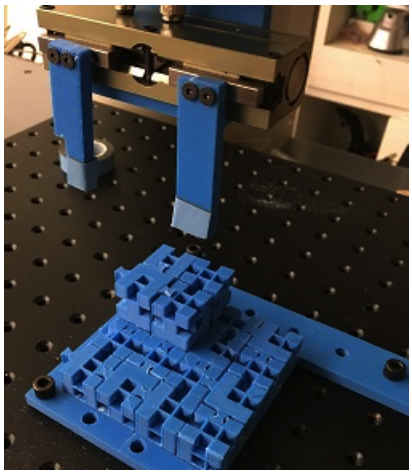


Fig. 11: Robot arm assembling a 2-layer structure

Table II shows number of blocks, layers and running time for executions of the design algorithm for various structures. Figure 11 shows a robot arm assembling a two layer structure with pure translations.

Model	# blocks	# layers	Timing (ms)
alpaca	400	18	15
pyramid	552	2	15
elephant	784	12	22
hollow cube	2184	8	69
hollow cube	3264	12	100
solid cube	4400	20	132
solid cube	6600	30	201
solid cube	8360	38	263
solid cube	9680	44	316
house	13104	40	633

TABLE II: Statistics about some structures

X. Conclusions and Future Work

This paper explores a solution to assemble general voxelized models as interlocking structures. Our method automatically generates a set of block types, assembly order and assembly directions that form the shape of the input voxel model.

Future work will include further experiments in building real-world models based on the design algorithm. In the experiment shown, joints have been filed to allow easier assembly, at the cost of some structural rigidity. We intend to explore alternate construction techniques that allow much larger models. We expect some portions of the assembly can be done in parallel. For example, some segments can be assembled mostly independently.

We would like to avoid the step in the algorithm in which some squares are removed from an odd layer. Choosing different assembly orders, patterns for laying out the design, or new block types might allow a greater number of completely filled voxels. We would also like to minimize the number of keys used. For the example models we considered, the number of keys is small, but models with layers that are not well-covered by upper-adjacent layers may have many keys using our approach.

We also wonder if the nine block types represent a lower bound; if similar techniques can be used with fewer block types, both production of block types and assembly might be simplified.

References

[1] DJ Arbuckle and Aristides AG Requicha. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Autonomous Robots*, 28(2):197–211, 2010.

[2] W. H. Culter. A computer analysis of all 6-piece burrs. *Self published*, 1994.

[3] Herbert Freeman and L Garder. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *Electronic Computers, IEEE Transactions on*, (2):118–127, 1964.

[4] Chi-Wing∗ Fu, Peng∗ Song, Xiaoqi Yan, Lee Wei Yang, Pradeep Kumar Jayaraman, and Daniel Cohen-Or. Computational interlocking furniture assembly. *ACM Transactions on Graphics (SIGGRAPH 2015)*, 34(4):091:1–091:11, 2015. ∗ joint first author.

[5] David Goldberg, Christopher Malon, and Marshall Bern. A global approach to automatic solution of jigsaw puzzles. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 82–87. ACM, 2002.

[6] Xu Minggang Qiu Hongxing. Analysis of seismic characteristics of chinese ancient timber structure.

[7] Weixin Kong and Benjamin B Kimia. On solving 2d and 3d puzzles using curve matching. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–583. IEEE, 2001.

[8] KENGO KUMA and ASSOCIATES. Prostho museum research center. 2010.

[9] Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. Converting 3d furniture models to fabricatable parts and connectors. *ACM Trans. Graph.*, 30(4):85:1–85:6, July 2011.

[10] Kui-Yip Lo, Chi-Wing Fu, and Hongwei Li. 3D Polyomino puzzle. *ACM Tran. on Graphics (SIGGRAPH Asia)*, 28(5), 2009. Article 157.

[11] Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. Chopper: Partitioning models into 3D-printable parts. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 31(6), December 2012.

[12] George A Popescu, Tushar Mahale, and Neil Gershenfeld. Digital materials for digital printing. In *NIP & Digital Fabrication Conference*, volume 2006, pages 58–61. Society for Imaging Science and Technology, 2006.

[13] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[14] Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[15] Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. Sketchchair: an all-in-one chair design system for end users. In *Proceedings of the 5th International Conference on Tangible and Embedded Interaction 2011, Funchal, Madeira, Portugal, January 22-26, 2011* [15], pages 73–80.

[16] Peng Song, Chi-Wing Fu, and Daniel Cohen-Or. Recursive interlocking puzzles. *ACM Transactions on Graphics (SIGGRAPH Asia 2012)*, 31(6):128:1–128:10, December 2012.

[17] Peng Song, Zhongqi Fu, Ligang Liu, and Chi-Wing Fu. Printing 3d objects with interlocking parts. *Computer Aided Geometric design (Proc. of GMP 2015)*, 35-36:137–148, 2015.

[18] Shiqing Xin, Chi-Fu Lai, Chi-Wing Fu, Tien-Tsin Wong, Ying He, and Daniel Cohen-Or. Making burr puzzles from 3D models. *ACM Transactions on Graphics (SIGGRAPH 2011 issue)*, 30(4):97:1–97:8, August 2011.

[19] K. Zwerger and V. Olgiati. *Wood and Wood Joints: Building Traditions of Europe, Japan and China*. Birkhäuser, 2012.