

# Recursive Interlocking Puzzles

Peng Song      Chi-Wing Fu  
Nanyang Technological University

Daniel Cohen-Or  
Tel Aviv University

## Abstract

Interlocking puzzles are very challenging geometric problems with the fascinating property that once we solve one by putting together the puzzle pieces, the puzzle pieces interlock with one another, preventing the assembly from falling apart. Though interlocking puzzles have been known for hundreds of years, very little is known about the governing mechanics. Thus, designing new interlocking geometries is basically accomplished with extensive manual effort or expensive exhaustive search with computers.

In this paper, we revisit the notion of interlocking in greater depth, and devise a formal method of the interlocking mechanics. From this, we can develop a constructive approach for devising new interlocking geometries that directly guarantees the validity of the interlocking instead of exhaustively testing it. In particular, we focus on an interesting subclass of interlocking puzzles that are recursive in the sense that the assembly of puzzle pieces can remain an interlocking puzzle also after sequential removal of pieces; there is only one specific sequence of assembling, or disassembling, such a puzzle. Our proposed method can allow efficient generation of recursive interlocking geometries of various complexities, and by further realizing it with LEGO bricks, we can enable the hand-built creation of custom puzzle games.

**CR Categories:** J.6 [Computer Applications]—Computer-aided design; K.8 [Personal Computing]: Games

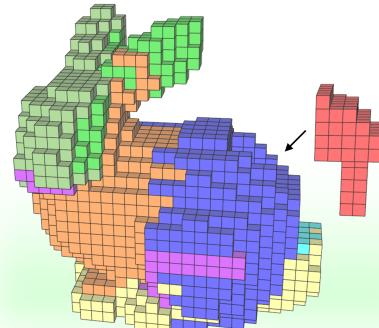
**Keywords:** Computer-aided design, interlocking, 3D puzzles

**Links:** [DL](#) [PDF](#)

## 1 Introduction

3D puzzles are generally nontrivial geometric problems that challenge our ingenuity. They have been longstanding stimulating recreational artifacts, where the task is to put together the puzzle pieces to form a meaningful 3D shape. Such an endeavor requires spatial cognitive skills for recognizing and understanding patterns and 3D structures, as observed from different angles. Interestingly enough, while solving a 3D puzzle assembly problem is already an intricate task, the creation of nontrivial puzzles is an even greater challenge.

Among the various families of 3D puzzles, a particularly challenging case is interlocking puzzles, for which one has to identify and follow certain orders to assemble the puzzle pieces into the target shape. The fascinating property of interlocking puzzles is that once



**Figure 1:** A 10-piece recursive interlocking BUNNY puzzle, where the red piece is the first key in the puzzle.

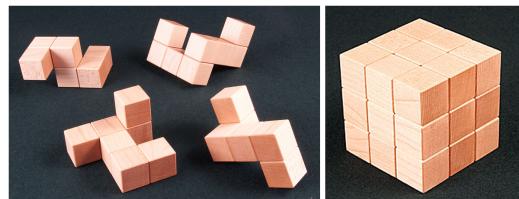
they are assembled, the puzzle pieces interlock with one another, preventing the 3D assembly from falling apart. More precisely, all puzzle pieces become immobilized except one single key piece, which is the last puzzle piece inserted to the assembly, and also the first required to be removed in the disassembly.

The geometric properties of interlocking puzzles are highly intriguing, and have been a virtue attracting architecture designers because the structure remains steady and solid while being glue-less, nail-less, and screw-less, and yet can be repeatedly assembled and disassembled. Nevertheless, little is known about the governing mechanics of interlocking puzzles [Cutler 1994; IBM Research 1997; Xin et al. 2011], and we are unaware of any previous research work in various communities, including graphics, computer aided design, and computational geometry. In this paper, we revisit the notion of interlocking in greater depth and develop a computational method for creating interlocking puzzles of varying complexities.

We define an interlocking puzzle as follows:

An assembly of puzzle pieces (with at least three pieces) is said to be interlocking if there exists only one movable puzzle piece, while all other puzzle pieces, as well as any subset of the puzzle pieces, are immobilized relative to one another.

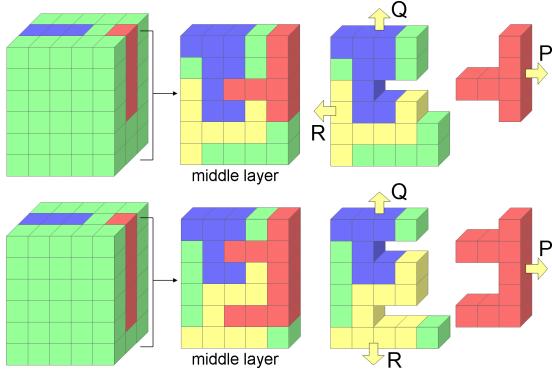
This definition stresses the difficulty of creating interlocking structures because, geometrically, the immobilization of individual puzzle pieces does not imply the immobilization of subsets of puzzle pieces. Thus, a naive decomposition of a shape into interlocking pieces would require (i) testing the immobilization of all the subsets of pieces, and (ii) testing that the decomposition is not deadlocked. Such tests lead to extremely expensive algorithms. Hence, previous attempts to discover new interlocking puzzles based on exhaustive search [Cutler 1978; Cutler 1994; IBM Research 1997] are extremely expensive. Other attempts, which are created manually, are very limited and still require enormous effort.



**Figure 2:** Coffin's four-piece interlocking cube.

Nevertheless, a manually-designed celebrated example is the interlocking cube of Coffin, shown in Figure 2. Unlike common burr interlocking structures [IBM Research 1997] whose mechanics, or blocking geometries, are centralized in a small region in the middle of the burr structure, here the interlocking geometry is decentralized, and the blocking parts in each puzzle piece are not as obvious as that in the burr structure. As one can observe in Figure 2, all the puzzle pieces appear structure-less, none suggesting its orientation with respect to the source shape, and yet, they can be assembled.

Coffin’s interlocking cube motivated us to study the geometric mechanics that govern, or produce, interlocking. We devise a formal method to describe and analyze the interlocking mechanics, and then develop a *constructive* approach for devising new interlocking geometries. Our method is constructive in the sense that it directly guarantees the validity of the interlocking instead of exhaustively testing it.



**Figure 3:** Examples. Top: recursive interlocking. After  $P$  is taken out, we must take out  $Q$  in order to take out  $R$ . Bottom: interlocking but not recursive interlocking. After  $P$  is taken out, we can take out either  $Q$  or  $R$ . Note that we assume when we remove a puzzle piece, we move it all the way out rather than moving it partially.

In our work, we focus on an interesting subclass of interlocking puzzles that are *recursive* in the sense that the assembly of puzzle pieces (with at least three pieces) remains an interlocking puzzle also after the (sequential) removal of pieces (see Figure 3). Thus, the disassembly (and assembly) of a recursive interlocking puzzle has a unique solution. In other words, there is a specific sequence of pieces with which the shape can be assembled (and disassembled).

Our formal derivation of recursive interlocking shows that if we maintain certain interlocking properties locally within every subsequent group of puzzle pieces in the puzzle disassembly sequence, the entire puzzle is proved to be interlocking. In this way, we can avoid the expense of checking the required interlock immobilization over all subsets of puzzle pieces. As a result, we can efficiently create larger interlocking structures.

We show that our constructive method allows generating new recursive interlocking structures of various complexities. In particular, we succeed in quickly generating a 20-piece  $6^3$  CUBE, which is larger than a 19-piece example (manual effort, just interlocking) we are aware of. We also succeed in generating 1250 pieces for a  $35^3$  CUBE, a task that is intractable by manual effort. Finally, like common interlocking structures that make up of cubical shapes, our method takes general voxelized shapes as input. Thus, as we show, the puzzles can be realized with physical LEGO bricks, offering simple means for the creation of “home-made” 3D puzzles.

## 2 Related work

**Computational Methods for Physical Models.** In recent years

various computational methods have been introduced for creating 3D structures that can be physically constructed. Mitani et al. [2004] approximated a given surface mesh by triangle strips to support the construction of paper-craft models. Kilian et al. [2008] proposed a method to construct developable surfaces with curved folding by minimizing the bending energy. Li et al. [2010] developed an automatic method to generate 3D pop-up paper architectures from 3D models based on a planar layout formulation. Tachi [2010] proposed a two-step mapping algorithm to origamize a 3D shape by producing a planar crease pattern that can be folded to form a 3D shape without any cut. More recently, Li et al. [2011] developed an interactive tool enabled by an automatic construction method for designing v-style pop-up cards, while Hildebrand et al. [2012] developed new methods to generate cardboard sculptures that can be fabricated from planar slices.

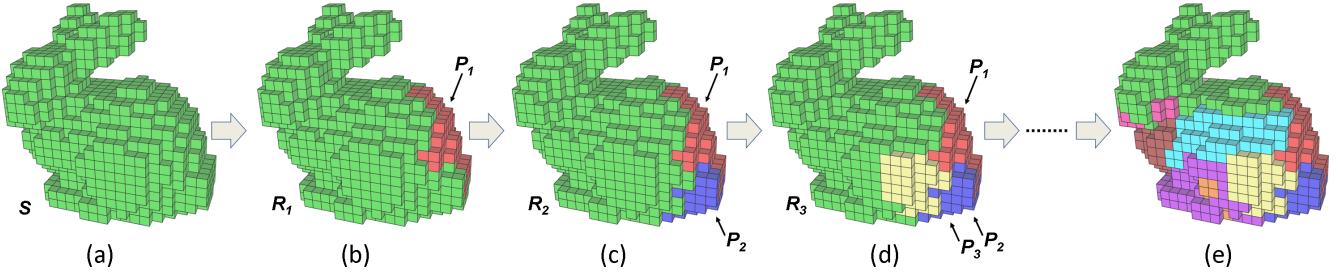
Computational methods were also devised to support physical construction of various other types of designs. Mori and Igarashi [2007] developed an interactive sketch interface that supports the design of 3D Plush toys from 2D cloth patterns. Weyrich et al. [2007] proposed a pipeline for creating bas-relief sculptures that meet various unique requirements such as depth discontinuity. Mitra and Pauly [2009] designed a computational framework for creating 3D shadow art by analyzing the constraints among shadow shapes cast in different directions. Alexa and Matusik [2010] proposed a sub-pixel representation for constructing reliefs so that a relief can appear to show different pictures when illuminated with different lighting directions. Most recently, Holroyd et al. [2011] devised a pipeline of computational methods that can avoid visual cracks when converting a 3D model into a multilayer artifact, while Lau et al. [2011] converted furniture models into fabricatable parts and connectors using lexical and structural analysis.

Computational methods for puzzles have received special attention. Early work focused mainly on how to apply computers to solve puzzle problems: 2D apictorial jigsaw puzzles [Freeman and Gardner 1964; Wolfson et al. 1988; Goldberg et al. 2002], 2D pictorial jigsaw puzzles [Murakami et al. 2008; Cho et al. 2010], and 2D/3D fragment assembly [Kong and Kimia 2001; Sagiroglu and Ercil 2006]. Recently, geometric methods have been proposed to generate 3D puzzle problems, including the work of Lo et al. [2009], who created shell-based 3D puzzles with polyominoes as the component shape of the puzzle pieces, and the work of Xin et al. [2011], who replicated and connected pre-defined six-piece burr structures to create interlocking puzzles from 3D models.

**Creating Interlocking Puzzles.** In this paper, we are interested in developing computational methods to generate novel, rather than replicating, interlocking structures.

Interlocking has a long history in the making of wooden artifacts. In ancient Chinese and Japanese architecture [Seike 1977; Zwerger 2012], the wooden joint on top of a pillar actually contains a small wooden pin (key) that can lock all local parts around it, including the crossbeams and decorations. With such interlocking, the overall timber framework is more stable and durable.

The design of a new interlocking configuration is extremely hard for humans, even for skilled professional. This perhaps explains why over many centuries there are not many known interlocking puzzles, and that these puzzles were designed by skillful craftsmen with great effort. As mentioned in [Coffin 1990], interlocking puzzle design requires hours, or even days, of mental work. Only in the late 1970s, Cutler [1978; 1994] proposed to use computers to exhaustively try and discover new possible configurations, thus introducing a large number of six-piece interlocking structures. Nevertheless, due to the combinatorial complexity, his computer program took almost three years to loop over the tens of billions of



**Figure 4:** Overview: (a) a voxelized shape as input,  $S$ ; (b) construct the key piece  $P_1$  (with remaining volume  $R_1$ ); (c) construct the next piece  $P_2$  (with remaining volume  $R_2$ ); and (d,e) iteratively construct all other pieces.

possible configurations where each configuration involves only six puzzle pieces and a variable cubical volume of less than  $4^3$  voxels [IBM Research 1997]. Other than complete analysis of all possible configurations, which is practical only for very small problem sets, some 3D puzzle designers took a trial-and-error approach by using computer software such as BurrTools by Röver [2011] as a puzzle solver to test if their puzzle designs can be assembled. Although this approach saves hours of time with papers and pencils, the design of interlocking configurations remains highly dependent on human intelligence, and yet only for small-scaled problems.

Moving forward in the holy grail of interlocking, a very recent work by Xin et al. [2011] attempted to create larger interlocking puzzles by replicating and connecting a known six-piece burr puzzle. Since it is based on reusing a typical burr puzzle, this method does not really create new interlocking configurations. In contrast, we first analyze the interlocking mechanics, and through this study, new interlocking configurations are created. In addition, compared to our method, which is fully automatic, Xin et al. [2011]’s method requires one to first manually design and construct a grid-based graph inside a given 3D shape, so that the method can put a six-piece burr puzzle at each grid point and connect them to guarantee the interlocking. Furthermore, since each burr puzzle at the graph grid points actually requires a bulky centralized structure in the middle to achieve interlocking, this method is less flexible when dealing with complex shapes and topologies. Unlike Xin et al.’s method, we start by studying the mechanics of interlocking in greater depth and devising a formal method on interlocking. Hence, we can develop an efficient constructive method capable of devising new interlocking geometries, and directly guarantee the validity of the interlocking rather than exhaustively testing it. Moreover, since our work follows the mechanism of Coffin’s interlocking cube, unlike burr puzzles, the interlocking structure is decentralized, and thus can adapt more flexibly to complex 3D shapes and topologies.

### 3 Overview

Figure 4 outlines in high-level our computational method. Like common interlocking structures, our method takes a general voxelized shape, denoted  $S$ , as input (Figure 4(a)), and iteratively extracts pieces, one by one, forming a sequence of extracted pieces  $P_1, P_2, \dots, P_n$ , with  $R_n$ , the remaining part of  $S$ , as the last piece:

$$S \rightarrow [P_1, R_1] \rightarrow [P_1, P_2, R_2] \rightarrow \dots \rightarrow [P_1, \dots, P_n, R_n].$$

The union of all resulting pieces is  $S$ , and our goal in this iterative process is to ensure that any (postfix) subsequence  $[P_i, \dots, P_n, R_n]$ , for  $i = 1..n - 1$ , is a recursive interlocking puzzle, where  $P_i$  is the key and all the other pieces in the subsequence (and any subset of it) are immobilized. To facilitate the understanding, we use a consistent color scheme on the following puzzle pieces: red for  $P_1$ , blue for  $P_2$ , yellow for  $P_3$ , orange for  $P_4$ , and green for the remaining volume from which the next piece is to be extracted (see again Figure 4).

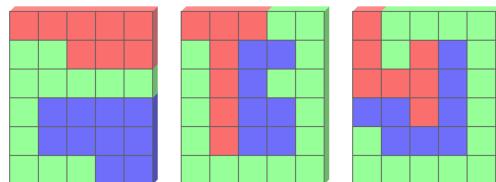
To achieve this goal, during the iterative extraction of puzzle pieces, we require a local interlocking among  $P_i$ ,  $P_{i+1}$ , and the remaining part  $R_{i+1}$ , e.g.,  $P_1$ ,  $P_2$ , and  $R_2$  are interlocking with  $P_1$  as the key (Figure 4(c)). Such local interlocking can be enforced by requiring that (i)  $P_i$  is the key piece in the local configuration  $[P_i, P_{i+1}, R_{i+1}]$ , while all other pieces and subsets of this configuration are immobilized; and (ii) once  $P_i$  is removed,  $P_{i+1}$  can be removed from  $R_{i+1}$  (see Section 4 for detail).

In Section 5, we describe how to extract  $P_i$ ’s to guarantee the above properties. The key idea is that by enforcing local interlocking among small subsequences, it can be proved by induction (Appendix) that puzzle pieces carefully constructed by our rules are guaranteed to be interlocked without exhaustive interlocking tests globally over subsets of all puzzle pieces. Moreover, such constructive requirement can be realized by an efficient method that examines, and generates, the blocking mechanics only locally (Section 5), allowing the efficient construction of interlocking structures with large number of puzzle pieces.

Once we define the puzzle pieces (Figure 4(e)), we hand-build the interlocking puzzles from standard LEGO bricks (Section 6).

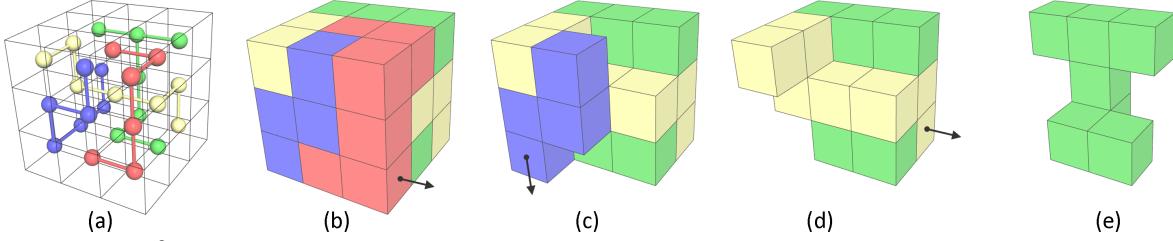
### 4 Formal Method

The decomposition of a 3D solid into pieces may yield either an *assemblable* or *non-assemblable* configuration. An *assemblable* decomposition can either be *interlocking* or *non-interlocking*. See Figure 5 for 2D examples. However, note that in 2D, a piece can be moved only by shifting it along the main axes of the plane, and removing a piece “vertically” up is an illegal move. In any case, our interest is in 3D where the pieces can only be moved by translating them along one of the three main axes. It should be noted that once a piece is translated by moving along a certain axis and becomes removable, we will remove it completely; we do not consider small and partial translations.



**Figure 5:** Decomposing a solid in 2D space into non-interlocking, interlocking, and non-assemblable pieces (left to right).

A non-interlocking configuration is one where more than one piece is mobilized. In Figure 5(left), the three pieces are all removable (in any order). A decomposition can be carefully defined so that the pieces block one another and there is only a single piece (the key) that is mobilized (see Figure 5(middle)). However, there could be an excess of blocking among the pieces (see Figure 5(right)), leading to a deadlock, and the decomposition cannot be assembled



**Figure 6:** A 4-piece  $3^3$  recursive interlocking CUBE generated by our method: (a) the puzzle piece anatomy; and (b-e) its disassembly.

nor disassembled. To achieve a valid interlocking in a 3D solid decomposition, we have to make sure that the decomposition is interlocking and that it can be assembled. Furthermore, to achieve recursive interlocking, we have to ensure that all intermediate configurations during the assembly (and disassembly) are properly interlocked. See Figure 6 for an example recursive interlocking CUBE.

In the following, we start by presenting our requirements that lead to constructing recursive interlocking puzzle pieces.

#### 4.1 Requirements on Constructing the Puzzle Pieces

To achieve local interlocking that can be guaranteed to lead to global (recursive) interlocking among all constructed puzzle pieces, we propose the following requirements on local interlocking (see again our notations on puzzle pieces in Section 3):

**Requirements on constructing  $P_1$ .** When we decompose an input 3D solid into  $P_1$  and  $R_1$ , we have the following requirements:

1. First, we can remove  $P_1$  directly in the two-piece configuration  $[P_1, R_1]$  with a single-step one-dimensional translation. This is to ensure that  $P_1$  can always be removed in the first step in the recursive interlocking puzzle;
2. Second,  $P_1$  should be removable in *only* one direction. If  $P_1$  can be removed in more directions, it may fall off easily, as well as leave fewer (or no) choices for the next piece (By Lemma 3, see Appendix);
3. Lastly,  $P_1$  should be simply connected. In addition, the same goes to  $R_1$  too; otherwise, we cannot enforce recursive interlocking for the remaining puzzle pieces. This is required for recursive interlocking, but not for simple interlocking.

**Requirements on constructing  $P_i$  ( $i > 1$ ).** After constructing the key,  $P_1$ , we can iteratively extract pieces one by one from the remaining volume with the following requirements:

1. First, the three-piece configuration  $[P_{i-1}, P_i, R_i]$  should be interlocking with  $P_{i-1}$  as the key. This local interlocking criteria can be proved to be fulfilled (by Lemma 2 in Appendix) by ensuring that (i)  $P_i$  is immobilized in configuration  $[P_{i-1}, P_i, R_i]$  and (ii)  $P_{i-1}$  and  $P_i$  cannot move together relative to a fixed  $R_i$ ;
2. Second, we have to make sure that  $P_i$  can be removed and separated from  $R_i$  in the two-piece configuration  $[P_i, R_i]$  after  $P_{i-1}$  is removed;
3. Lastly, like the last requirement on  $P_1$ , both  $P_i$  and  $R_i$  should be simply connected.

**Global interlocking.** The above requirements actually lead to a formal model, and later our constructive method, which iteratively extracts subsequences of locally-interlocked puzzle pieces. In particular, we found that by using mathematical induction (see Appendix), it is possible to show that puzzle pieces constructed with these requirements can be guaranteed to be recursive interlocking, regardless of the number of puzzle pieces involved. Thus, global interlocking can be directly guaranteed without exhaustive tests.

## 5 Our Constructive Approach

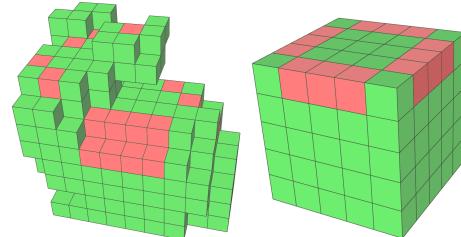
Our constructive approach for devising new interlocking geometries has two major procedures: (i) extract the key piece from the input volume, and (ii) iteratively extract other puzzle pieces one by one. Various blocking and unblocking mechanics are explored in these two procedures in order to construct the puzzle pieces that meet the specified requirements.

Let us first denote  $N$  as the total number of voxels representing the given model and  $K$  the number of puzzle pieces to be constructed. To balance the size of the puzzle pieces, our method attempts to construct each piece with roughly  $m = \lceil N/K \rceil$  voxels, except for the primary key  $P_1$ , which could be rather small.

### 5.1 Extracting the Key Piece

The procedure for extracting the key piece includes:

- 1) **Pick a seed voxel.** We start by picking a seed voxel as a cornerstone for growing the key piece. First, we identify a candidate set of exterior voxels that have exactly a pair of adjacent exterior faces, with one being on the top, see Figure 7. Here we require axial free passages, that is, no voxels all the ways above these exterior faces. Hence, such voxel can be moved out in one translational step, which echoes the key's requirement. From the candidate set, we can either randomly pick a seed, or let the user make a choice. Moreover, we define *upward* as the default moving direction for the key, so that the assembled puzzle is more stable when sitting on a table in an intended display orientation.



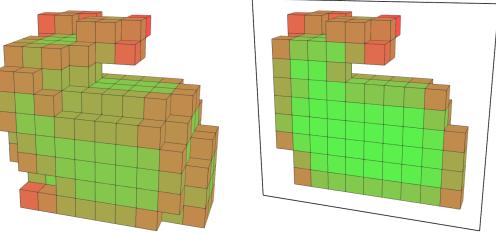
**Figure 7:** Candidate seed voxels (in red) should have exactly a pair of adjacent exterior faces with one being on the top.

- 2) **Compute voxel accessibility.** After extracting a puzzle piece, the remaining volume has to be connected. However, naive extraction of voxels may easily lead to fragmentation. Hence, we compute an accessibility value, say  $a_j(x)$ , for each voxel  $x$  in a remaining volume, and use it later as a heuristic to alleviate fragmentation, where  $a_j(x)$  is computed by recursively counting the (weighted) number of voxel neighbors:

$$a_j(x) = \begin{cases} \text{number of neighbors of } x, & \text{for } j = 0 \\ a_{j-1}(x) + \alpha^j \sum_i a_{j-1}(y_i(x)) & \text{for } j > 0, \end{cases}$$

where  $y_i(x)$ 's are neighboring voxels of  $x$  in the remaining volume. Note that the weight factor  $\alpha$  is set to 0.1 in our implementation. We stop the recursion at  $j = 3$  because we found experimentally

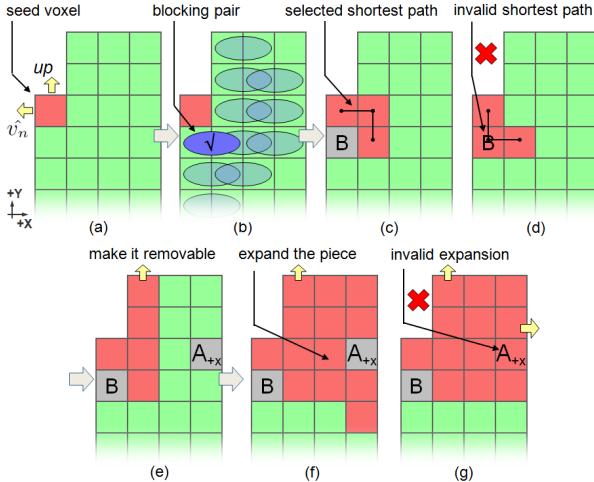
that the resulting accessibility values are sufficient for guiding the voxel selection, see Figure 8 for an example. Since voxels with low accessibility are likely to be fragmented, we prioritize to include them when constructing a puzzle piece.



**Figure 8:** Voxel accessibility on BUNNY. Left: red color indicates low accessibility while green color indicates high accessibility. Right: we show an internal slice by a clip plane.

**3) Ensure blocking and mobility.** Now, we are ready to develop the key piece such that it is removable by a translation along one direction. We have the following substeps (see Figure 9):

- First, we identify the normal direction, say  $\hat{v}_n$ , of the non-upward-facing exterior face of the seed voxel (Figure 9(a));
- Then, we do a breadth-first traversal from the seed to find  $N_{b1}$  pairs of voxels (that orient along  $\hat{v}_n$ ) that are the nearest to the seed (see the oval shapes in Figure 9(b)), where in each pair, the voxels on the positive and negative sides of  $\hat{v}_n$  are called the *blocking* and *blockee* voxels, respectively. Among them, we select  $N_{b2}$  pairs whose blockee has the smallest accessibility among the  $N_{b1}$  pairs (note: in our implementation,  $N_{b1}$  and  $N_{b2}$  are set to be 50 and 10, respectively);
- Next, we use the following three strategies for constructing the key with appropriate blockage:
  - First, we *block the key from moving towards  $\hat{v}_n$*  by (i) determining a set of shortest path candidates from the seed to each blockee voxel candidate (without crossing the related blocking voxel and voxels below it); we later will select one of them for evolving the key; and (ii) extracting all the voxels along a selected shortest path until the blockee, and adding these voxels to evolve key piece (Figure 9(b&c));
  - Second, we *ensure the key to be removable upward* by including any voxel above the selected shortest path (Figure 9(e)). This is why the shortest paths determined



**Figure 9:** Steps to develop the key piece from a seed voxel.

in the strategy above should not go through the blocking voxel or any voxel below it, else the blockage is destructed (Figure 9(d)). Moreover, we ignore the shortest path candidates that eventually add excessive voxels to the key since the key should have less than  $m$  voxels;

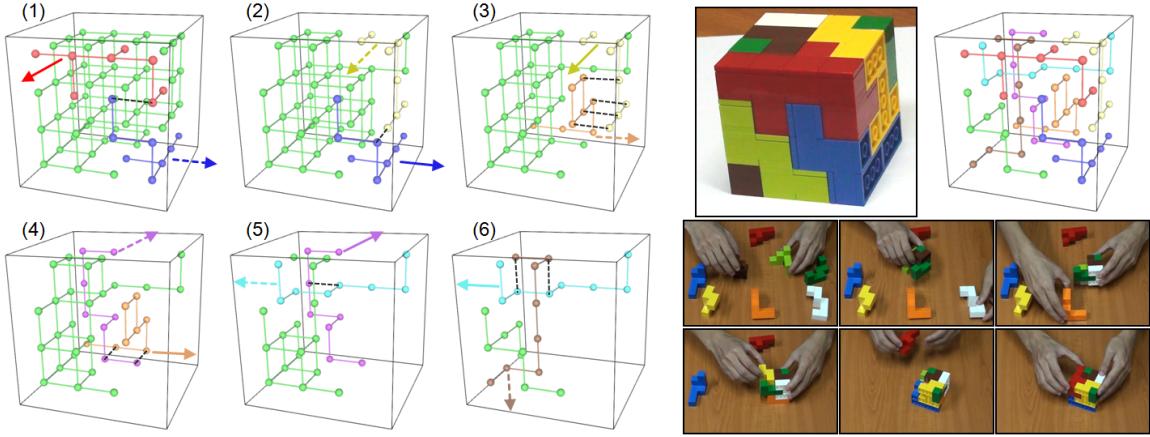
– So far, we can devise a key that moves upward but not along  $\hat{v}_n$ . However, since new voxels are added to the key, *the key may accidentally become mobilized* in a direction along which the seed was originally blocked, e.g.,  $+X$ ,  $-Y$ , and  $\pm Z$  for the seed in Figure 9. This would require testing the blockage (or mobility) every time we add voxels to the key. However, our third strategy avoids such test: we identify an *anchor voxel* that is directly-connected and furthest away from the seed along each initially-blocked direction of the seed, e.g., voxel  $A_{+x}$  for  $+X$  (Figure 9(e)). The key idea is that if these anchor voxels stay with the remaining volume (but not added to the key), the key can remain to be immobilized in the blocked directions even if we add more voxels to it (Figure 9(f)). Mobility test is not required to ensure the maintenance of the blockage.

- To choose among the shortest paths resulted from the first two strategies, we sum for each path the accessibility of all the voxels required to be added to the key, i.e., voxels along the path, the blockee, and any voxel above. Then, we pick the one with the smallest accessibility sum for evolving the key with the appropriate blockage.

**4) Expand the key piece.** Since the key piece usually has less than  $m$  voxels at this moment, the goal of step 4 is to augment it with more voxels to balance the size of the puzzle pieces:

- First, we identify an additional anchor voxel for the direction immobilized by the blocking voxel we picked in step 3. Like before, it is directly-connected and furthest away from the blocking voxel in direction  $\hat{v}_n$ ; if no such voxels exist, we use the blocking voxel as the anchor; Note that the anchor voxel idea is crucial for the expansion process as the existing blockage could be undesirably destructed if an anchor voxel is inappropriately added to the key piece (Figure 9(g)).
- Then, we identify a set of candidate voxels to be added to the key, say  $\{u_i\}$ , that are resided next to the key but neither at the anchor voxels nor below the anchors. For each  $u_i$ , we identify also the voxels directly above it, so that we know the voxels required to be added to the key if  $u_i$  is chosen. Furthermore, if the number of voxels exceeds the number of extra voxels the key needs, we remove  $u_i$  from the candidate set.
- Next, we sum the accessibility of each  $u_i$  and the voxels above it, say  $\text{sum}_i$ , and normalize  $p_i = \text{sum}_i^{-\beta}$  to be  $\hat{p}_i = p_i / \sum_i p_i$ , where  $\beta$  is a parameter ranged from 1 to 6. Hence, we can randomly pick a  $u_i$  with  $\hat{p}_i$  as the probability of choosing it, and expand the key piece. These substeps are repeated until the key contains roughly  $m$  voxels.

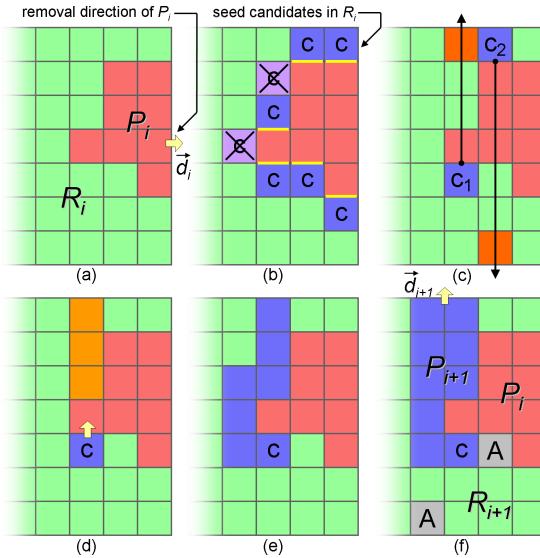
**5) Confirm the key piece.** After steps 1 to 4, the key is guaranteed to fulfill all interlocking requirements, except that  $R_2$  needs to be simply connected. With the help of accessibility, the chance of fragmenting  $R_2$  (and other remaining volumes) is rather low; hence, testing the connectivity of voxels in  $R_2$  at the end of the key piece generation procedure is more efficient than doing it at multiple places. To guarantee that  $R_2$  is simply connected, we gather all the voxels next to the key in a set, say  $R_s$ , and apply a simple flooding algorithm to test whether all voxels in  $R_s$  can be visited or not in  $R_2$ .



**Figure 10:** Left: local interlocking established among every three consecutive puzzle pieces in an 8-piece  $4^3$  CUBE; in each subfigure, the solid and dashed arrows show the moving direction of the current movable puzzle piece and its successive piece, respectively; the blockage between these two pieces is indicated by dashed black line(s). Right: our home-made  $4^3$  LEGO CUBE, its anatomy, and the building sequence.

## 5.2 Extracting Other Puzzle Pieces

Similar to that of the primary key piece, the procedure of extracting subsequent puzzle pieces, e.g.,  $P_{i+1}$  from  $R_i$ , also starts by picking a seed voxel, and then growing  $P_{i+1}$  from it. However, since there are additional requirements for local interlocking among  $P_i$ ,  $P_{i+1}$ , and  $R_{i+1}$ , the blocking mechanics are more involved. To facilitate our discussion, we denote  $\vec{d}_i$  as the target moving direction of  $P_i$ .



**Figure 11:** Steps to extract  $P_{i+1}$  from  $R_i$ .

**1) Candidate seed voxels.** Since  $P_{i+1}$  is blocked by  $P_i$ , but becomes mobilized as soon as  $P_i$  is removed, at least one of its voxel must reside next to  $P_i$ . Since successive puzzle pieces should move in different directions (by Lemma 3 in Appendix), we use the contact between  $P_i$  and  $P_{i+1}$  to define  $\vec{d}_{i+1}$  for blocking  $P_{i+1}$  by the presence of  $P_i$ . Our strategy is to pick voxels (in  $R_i$ ) next to  $P_i$  as candidate seeds, requiring them to contact  $P_i$  in a direction perpendicular to  $\vec{d}_i$ . See Figure 11(b) for valid and invalid candidates in blue and violet, respectively.

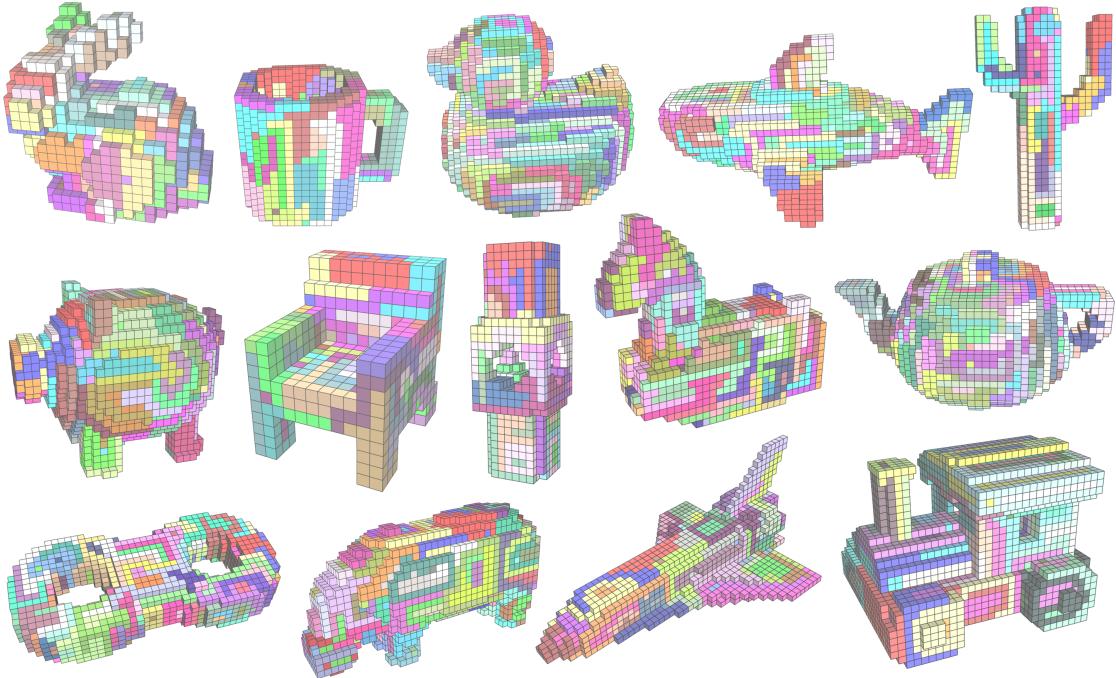
Since there may be too many valid candidates, trying them all is overly time consuming. Hence, we compute the accessibility of voxels in  $R_i$  and reduce the number of candidates to ten by the following equally-weighted criteria: (i) smaller accessibility value;

and (ii) shorter distance to the furthest-away voxel in  $R_i$  along  $\vec{d}_{i+1}$ , see Figure 11(c) for examples: an initial  $P_{i+1}$  formed by  $C_2$  will contain more voxels as compared to  $C_1$  because of a longer shortest path determined by step 2 below. Hence, the second criteria helps reduce the number of voxels that are required to form an initial  $P_{i+1}$ . Note that we attempt to use fewer voxels (in early steps) to construct an initial  $P_{i+1}$  because this allows us to have more flexibility when expanding the puzzle piece in step 3.

**2) Create an initial  $P_{i+1}$ .** After step 1, we have a set of candidate seeds, each associated with a  $\vec{d}_{i+1}$ . Our next step is to pick one of them by examining its cost of making  $P_{i+1}$  removable in  $\vec{d}_{i+1}$ : (i) from each candidate, we identify all voxels in  $R_i$  along  $\vec{d}_{i+1}$  (see the orange voxels in Figure 11(d)) since these voxels must be taken to  $P_{i+1}$  to make the candidate removable along  $\vec{d}_{i+1}$ ; (ii) we determine a shortest path to connect the candidate to these identified voxels (Figure 11(e)), and (iii) we locate also any additional voxel required to mobilize the shortest path towards  $\vec{d}_{i+1}$  (Figure 11(f)). To choose among the candidates, we sum the accessibility of all the voxels involved in each candidate path (blue voxels in Figure 11(f)), and pick the one with the smallest sum for forming the initial  $P_{i+1}$ .

**3) Ensure local interlocking.** Until now,  $P_{i+1}$  is modeled with appropriate blocking for direction  $\vec{d}_{i+1}$  (Figure 11(f)), where its mobility towards  $\vec{d}_{i+1}$  depends on the presence of  $P_i$  alone. Next, we further have to ensure appropriate blocking for the other five directions for achieving local interlocking in  $[P_i, P_{i+1}, R_{i+1}]$ : (i)  $P_{i+1}$  is immobilized in the presence of  $R_{i+1}$  and  $P_i$ , and (ii) it cannot co-move with  $P_i$ . For this, we perform a mobility check for each of the five directions to see if  $P_{i+1}$  is blocked or not. Note that the mobility check of  $P_{i+1}$  along any direction  $\vec{d}$  is done by checking if any voxel from  $P_i$  or  $R_{i+1}$  contacts  $P_{i+1}$  along  $\vec{d}$ . If this is true,  $P_{i+1}$  is immobilized to move along  $\vec{d}$ . Only if  $P_{i+1}$  is movable, we apply the first two strategies in Section 5.1 (step 3) to extend  $P_{i+1}$  to some blockee voxels for achieving appropriate blocking in related direction(s).

For requirement (i) above, we perform the mobility check on  $P_{i+1}$  in the presence of both  $P_i$  and  $R_{i+1}$ . However, the tricky part for direction  $\vec{d}_i$  is that since  $P_{i+1}$  should not be co-movable with  $P_i$ , we have to perform the mobility check on  $P_{i+1}$  in the absence of  $P_i$  for this particular direction. Lastly, note further that the anchor voxel strategy in Section 5.1 can also be applied here. See Figure 10 (left) for the local interlocking established in an 8-piece  $4^3$  CUBE.



**Figure 12:** Recursive interlocking puzzles. From left to right and then top to bottom: BUNNY, MUG, DUCK, SHARK, CACTUS, PIGGY, CHAIR, WORK PIECE, ISIDORE HORSE, TEAPOT, EIGHT, HIPPO, SHUTTLE, and TOY TRAIN.

**4) Expand  $P_{i+1}$  and 5) Confirm it.** After the above steps,  $P_{i+1}$  can fulfill the local interlocking requirement, but yet we have to expand it to  $m$  voxels and check whether  $R_{i+1}$  is simply connected or not. These are done in the same way as in Section 5.1.

## 6 Implementation and Results

**Implementation Issues.** To improve the computational efficiency for handling very large voxelized models, we localize the puzzle piece extraction process. We define a rectangular box with a margin of 10 voxels around the seed (for key) or the previous puzzle piece (for others), and then dynamically expand this box when growing a puzzle piece. Thus, computations, including accessibility, shortest paths, etc., are localized to the extent of the box.

**Results.** Our method can create recursive interlocking puzzles on voxelized models of various shapes and topologies, see Figure 12, e.g., EIGHT with two holes, MUG with a large open concave region in the middle, CACTUS with branches, and PIGGY, which is modeled as a coin bank with a central cavity and a coin slot on top. Our method can also work with models with large variation in voxel counts ( $N$ ) and average puzzle piece size ( $m$ ). Figure 6 presents our 4-piece recursive interlocking  $3^3$  CUBE, whereas Figure 14 presents our 1250-piece recursive interlocking  $35^3$  CUBE. All these puzzles are recursive interlocking. More examples are shown in Figure 15.

We believe that the 1250-piece CUBE is the largest (recursive) interlocking puzzle with the greatest number of puzzle pieces ever defined or known. The computation time to create it is around 10 hours. In contrast, the largest interlocking 3D puzzle we found in the Internet contains 19 puzzle pieces only, which is a  $6^3$  interlocking CUBE. Our method can also produce a  $6^3$  recursive interlocking CUBE, but with 20 pieces instead (and 22 if recursive interlocking is relaxed). Moreover, for a  $4^3$  interlocking CUBE, the largest number of puzzle pieces we aware of is 8, and our method can generate a  $4^3$  recursive interlocking CUBE with 8 pieces, see Figure 10 (right). However, since our CUBE puzzle is recursive, it is claimed to be a

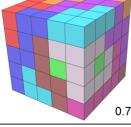
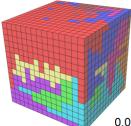
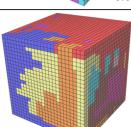
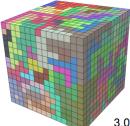
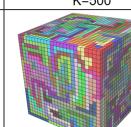
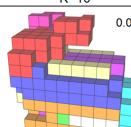
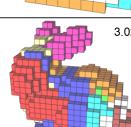
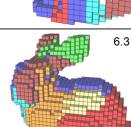
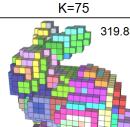
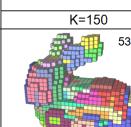
new puzzle that was not discovered before. Lastly, other than the  $4^3$  LEGO CUBE shown in Figure 10, we also create a LEGO BUNNY with 40 interlocking puzzle pieces and 966 voxels, see Figure 13.

**Performance.** Table 1 presents recursive interlocking CUBE and BUNNY in different  $N$ s (number of voxels) and  $K$ s (number of puzzle pieces). Note that this table is incomplete since puzzles with large  $K$  cannot be created with relatively small number of voxels. Timing statistics are also provided, showing that these puzzles can be created fairly efficiently with our method though the time taken to generate the puzzles could vary from seconds to hours depending on the the puzzle model. In more detail, the puzzle generation performance depends on several factors: a combination of  $N$  and  $K$ , as well as the complexity of the model's shape and topology:

- For fixed  $N$ , the computation time generally increases when  $m$  gets too small because for a small  $m$ , it is obviously harder to determine the shortest paths for constructing the interlocking geometry;
- On the other hand, the computation time also increases with  $N$  when  $K$  is fixed. Since each puzzle piece contains more



**Figure 13:** Our 40-piece home-made LEGO BUNNY.

Cube	K=10	
5x5x5		0.70 min
15x15x15		0.08 min
25x25x25		8.26 min
K=100		K=500
15x15x15		3.03 min
25x25x25		16.17 min
		110.70 min
Bunny	K=10	
10x9x7		0.03 min
20x19x15		3.02 min
30x29x23		6.31 min
K=75		K=150
		319.85 min
		78.71 min
		534.43 min

**Table 1:** Performance: generating recursive interlocking CUBE and BUNNY with different Ns and Ks.

voxels, it takes more time to find a suitable seed, to construct the shortest paths, and to expand the puzzle piece, etc.;

- Lastly, we also found that the computation time depends on the complexity of 3D shape, and shapes with branches and holes generally demand more computation time.

## 7 Conclusion

Interlocking is an intriguing but complex mechanical state, where assembled component pieces appear to lock one another. Yet, the puzzle can be disassembled through certain sequences of moves starting from the key. Rather than creating interlocking structures by manual effort, or by expensive exhaustive search with computers, we develop a novel computational method that enables the creation of new interlocking geometry by manipulating the underlying blocking mechanics. We focus on an interesting and challenging subclass of interlocking, called recursive interlocking, where all intermediate assembling (and disassembling) stages remain interlocking, resulting in a unique sequence of moves.

In summary, our contributions are as follow. As inspired by the way Coffin’s interlocking cubes work, we first derive a formal method of the problem of interlocking. After revisiting the notion of interlocking, we contrast interlocking against other scenarios of 3D solid decompositions, which are either not interlocking, or not capable of being assembled (deadlocking). Then, we formulate the requirements and derive a formal model for enforcing recursive interlocking, where the puzzle pieces constructed accordingly are guaranteed to be globally interlocked regardless of the number of puzzle pieces involved. These enable us to create recursive inter-

locking puzzles of various complexities, and to extend the number of interlocking puzzle pieces that one can ever construct. Second, based on the formal model, we devise a constructive approach that iteratively creates interlocking puzzle pieces. A family of construction strategies that manipulate the blocking mechanics is presented for producing appropriate blockage according to the requirements. Lastly, we also demonstrate the feasibility of creating home-made interlocking 3D puzzles with LEGO bricks.

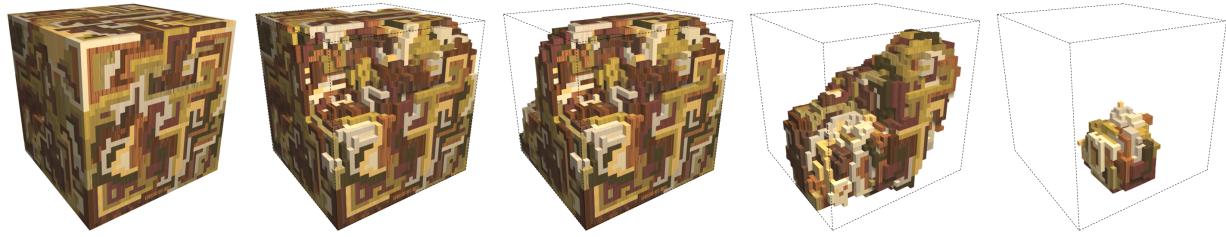
**Limitations.** First, we may not be able to generate interlocking puzzles with the largest possible  $K$  for a given voxelized shape, in particular when the shape is not so small. This is due to the fact that the solution space is astronomical since the number of possible ways to dissect an input model increases exponentially with the voxel count. Second, our method assumes no rotation of puzzle pieces. Some existing wooden puzzles have a small tricky step, requiring that a certain puzzle piece must be rotated to remove or assemble, see also Coffin’s book [1990]. Lastly, our method cannot handle input models with voxels connected by an edge rather than a face, and one-voxel-layer thin models, e.g., a  $1 \times 3 \times 10$  plane.

**Future work.** Our formal model can be generalized to create interlocking (but not recursive) puzzles by (i) allowing the decomposition of a remaining volume into three or more pieces at a time, and requiring that the decomposed pieces together with their preceding piece are locally interlocking; and (ii) allowing a remaining volume after puzzle piece extraction to be disconnected. Yet, we can modify the proof in the appendix to show that the resulting puzzle after this generalization can still be assembled into an interlocking structure (though no longer recursive). In this way, we can also avoid exhaustive testing on interlocking. Second, we plan to construct puzzle pieces that are geometry-aware. By identifying groups of voxels that contribute to different surface features on the 3D shape, we can prioritize the selection of groups of voxels altogether when making the puzzle pieces. Moreover, we could also symmetrize voxel groups if the input shape is symmetric. Lastly, to deal with more general shapes other than voxels, we plan to explore the space carving technique in Xin et al. [2011], i.e., the possibility of extruding our voxelized puzzle pieces to meet the general surface of the associated 3D shape.

**Acknowledgments.** We thank anonymous reviewers for the various constructive comments, John Rausch of [www.johnrausch.com](http://www.johnrausch.com) for sharing the photos shown in Figure 2, Michael Brown for narrating the video presentation, William Lai for his help on 3D Studio Max, and William Hutama for building the LEGO puzzle. This work is supported in part by the MOE Tier-2 grant (MOE2011-T2-2-041), Singapore, and the Israel Science Foundation.

## References

- ALEXA, M., AND MATSIK, W. 2010. Reliefs as images. *ACM Trans. on Graphics (SIGGRAPH)* 29, 4, Article 60.
- CHO, T. S., AVIDAN, S., AND FREEMAN, W. T. 2010. A probabilistic image jigsaw puzzle solver. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 183–190.
- COFFIN, S. T. 1990. *The Puzzling World of Polyhedral Dissections*. Oxford University Press.
- CUTLER, W. H. 1978. The six-piece burr. *Journal of Recreational Mathematics* 10, 4, 241–250.
- CUTLER, W. H., 1994. A computer analysis of all 6-piece burrs. Self published.



**Figure 14:** Disassembling our recursive interlocking  $35^3$  CUBE with 1250 pieces (now rendered in wooden style). From left to right, the number of puzzle pieces are 1250, 1150, 950, 350, and 50, respectively.

- FREEMAN, H., AND GARDER, L. 1964. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *IEEE Transactions on Electronic Computers EC-13*, 2, 118–127.
- GOLDBERG, D., MALON, C., AND BERN, M. 2002. A global approach to automatic solution of jigsaw puzzles. In *Proceedings of the Eighteenth Annual ACM Symposium on Computational Geometry*, 82–87.
- HILDEBRAND, K., BICKEL, B., AND ALEXA, M. 2012. crdbrd: Shape fabrication by sliding planar slices. *Computer Graphics Forum (Eurographics)* 31, 2, 583–592.
- HOLROYD, M., BARAN, I., LAWRENCE, J., AND MATUSIK, W. 2011. Computing and fabricating multilayer models. *ACM Trans. on Graphics (SIGGRAPH ASIA)* 30, 6. Article 187.
- IBM RESEARCH, 1997. The burr puzzles site. <http://www.research.ibm.com/BurrPuzzles/>.
- KILIAN, M., FŁÓERY, S., CHEN, Z., MITRA, N. J., SHEFFER, A., AND POTTMANN, H. 2008. Curved folding. *ACM Trans. on Graphics (SIGGRAPH)* 27, 3.
- KONG, W., AND KIMIA, B. B. 2001. On solving 2D and 3D puzzles using curve matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 583–590.
- LAU, M., OHGAWARA, A., MITANI, J., AND IGARASHI, T. 2011. Converting 3d furniture models to fabricatable parts and connectors. *ACM Trans. on Graphics (SIGGRAPH)* 30, 4. Article 85.
- LI, X.-Y., SHEN, C.-H., HUANG, S.-S., JU, T., AND HU, S.-M. 2010. Popup: Automatic paper architectures from 3D models. *ACM Trans. on Graphics (SIGGRAPH)* 29, 3. Article 111.
- LI, X.-Y., JU, T., GU, Y., AND HU, S.-M. 2011. A geometric study of  $v$ -style pop-ups: theories and algorithms. *ACM Trans. on Graphics (SIGGRAPH)* 30, 4. Article 98.
- LO, K.-Y., FU, C.-W., AND LI, H. 2009. 3D Polyomino puzzle. *ACM Trans. on Graphics (SIGGRAPH Asia)* 28, 5. Article 157.
- MITANI, J., AND SUZUKI, H. 2004. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Trans. on Graphics (SIGGRAPH)* 23, 3, 259–263.
- MITRA, N. J., AND PAULY, M. 2009. Shadow art. *ACM Trans. on Graphics (SIGGRAPH Asia)* 28, 5. Article 156.
- MORI, Y., AND IGARASHI, T. 2007. Plushie: an interactive design system for plush toys. *ACM Trans. on Graphics (SIGGRAPH)* 26, 3. Article 45.
- MURAKAMI, T., TOYAMA, F., SHOJI, K., AND MIYAMICHI, J. 2008. Assembly of puzzles by connecting between blocks. In *19th International Conference on Pattern Recognition*, 1–4.
- RÖVER, A., 2011. Burr tools. <http://burrtools.sourceforge.net/>.
- SAGIROGLU, M., AND ERCIL, A. 2006. A texture based matching approach for automated assembly of puzzles. In *18th International Conference on Pattern Recognition*, vol. 3, 1036–1041.
- SEIKE, K. 1977. *Art Of Japanese Joinery*. Weatherhill.
- TACHI, T. 2010. Origamizing polyhedral surfaces. *IEEE Transactions on Visualization and Computer Graphics* 16, 2, 298–311.
- WEYRICH, T., DENG, J., BARNES, C., RUSINKIEWICZ, S., AND FINKELESTEIN, A. 2007. Digital bas-relief from 3D scenes. *ACM Trans. on Graphics (SIGGRAPH)* 26, 3. Article 32.
- WOLFSON, H., SCHONBERG, E., KALVIN, A., AND LAMDAN, Y. 1988. Solving jigsaw puzzles by computer. *Annals of Operations Research* 12, 51–64.
- XIN, S.-Q., LAI, C.-F., FU, C.-W., WONG, T.-T., HE, Y., AND COHEN-OR, D. 2011. Making burr puzzles from 3D models. *ACM Trans. on Graphics (SIGGRAPH)* 30, 4. Article 97.
- ZWERGER, K. 2012. *Wood and Wood Joints: Building Traditions of Europe, Japan and China*. Birkhäuser Verlag.

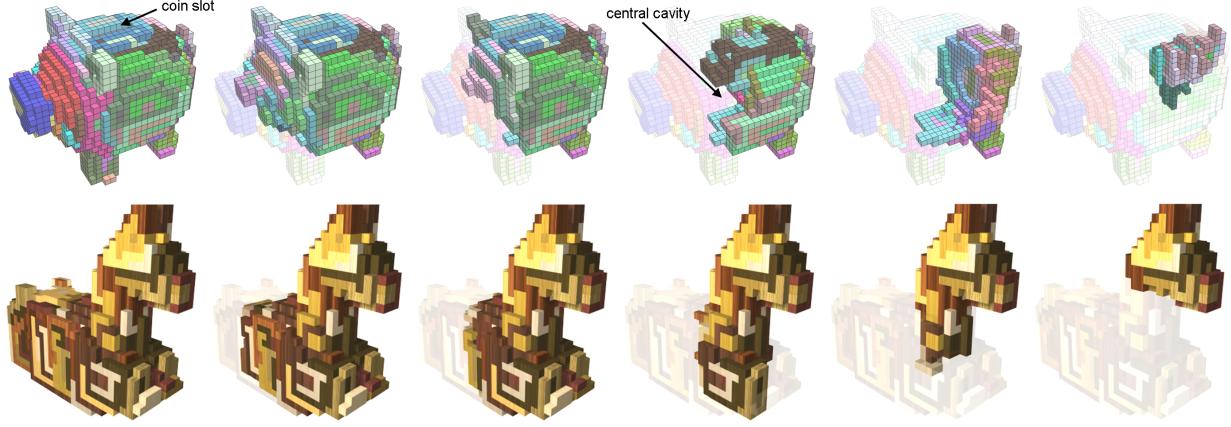
## Appendix A. Lemmas on Interlocking.

- **Lemma 1: Group Immobilization.** A group of puzzle pieces are immobilized for moving *together* in any direction if there is a puzzle piece (or a subset) in the group that is immobilized (blocked) by the complement pieces in the puzzle. Note that this lemma is also true for puzzle pieces that are not simply connected since they can still be moved together.
- **Lemma 2: Relativity Rule.** Given a set of assembled puzzle pieces that are divided into two non-empty sets,  $S_1$  and  $S_2$ :
  - If all puzzle pieces in  $S_1$  can move together in a certain direction  $D$  while  $S_2$  is fixed in space, then all puzzle pieces in  $S_2$  can also move together in opposite direction ( $-D$ ) while  $S_1$  is fixed in space;
  - The inverse of the above is also true: If  $S_1$  cannot move in a certain direction  $D$  (blocked by  $S_2$ ),  $S_2$  cannot move in  $-D$  (blocked by  $S_1$ ), and vice versa.
- **Lemma 3: Successive Moving Directions.** In a recursive interlocking puzzle, successive puzzle pieces should have different removal directions (in fact, perpendicular), else they can move together and violate the notion of interlocking.

These lemmas appear to be straightforward but as we shall soon see, they are helpful in deriving the formal model below.

## Appendix B. Proof: The Formal Model.

Here we prove by mathematical induction that puzzle pieces that are iteratively constructed according to the requirements in Section 4.1 are guaranteed to be recursive interlocking without exhaustively testing the subset immobilization.



**Figure 15:** Disassembling the PIGGY coin bank (top) and the ISIDORE HORSE puzzles (bottom).

Let  $C_n = [P_1, P_2, \dots, P_n, R_n]$  be the puzzle configuration after the extraction of  $n$  pieces. From the specified requirements, we can show that puzzle pieces in  $C_2$  ( $n = 2$ ) are capable of being assembled into a recursive interlocking structure [Q.E.D.].

Assume that  $C_n$  can be assembled into a recursive interlocking structure for some positive integer  $n \geq 2$ . Our goal is to show that if we base on our requirements to decompose  $R_n$  (in  $C_n$ ) into  $P_{n+1}$  and  $R_{n+1}$ , the next configuration  $C_{n+1}$  can also be assembled into a recursive interlocking structure.

For this, we have to prove the following three statements:

(i)  **$C_{n+1}$  can be assembled.** If a puzzle assembly can be assembled, it can also be disassembled. Therefore, we can prove this statement by showing that all pieces in  $C_{n+1}$  can be removed one after another. Since  $C_n$  can be assembled (and disassembled), we can sequentially remove  $P_1, P_2, \dots$ , up to  $P_n$  from  $C_{n+1}$  by regarding  $P_{n+1}$  and  $R_{n+1}$  of  $C_{n+1}$  as  $R_n$  of  $C_n$ . Then, by the requirement on  $P_i$ , the remaining two pieces, i.e.,  $P_{n+1}$  and  $R_{n+1}$ , can be disassembled too. Hence, if  $C_n$  can be disassembled (and assembled),  $C_{n+1}$  can also be disassembled (and assembled).

(ii)  **$C_{n+1}$  is interlocking.** This claim is more complicated than the above proof because we have to show that  $P_1$  is the *only* movable piece in  $C_{n+1}$ , while all other subsets of  $C_{n+1}$  (except  $P_1$  and  $C_{n+1} - P_1$ , by Lemma 2) are immobilized. For this, we need to prove the following three conditions:

1)  $P_1$  is removable in  $C_{n+1}$ . This can be proved by the requirement on the key piece  $P_1$  [Q.E.D.].

2) Every other piece ( $P_2, \dots, P_{n+1}$ , and  $R_{n+1}$ ) is immobilized:

- Since  $C_n$  is interlocking, each  $P_i$  ( $i = 2 \dots n$ ) in  $C_{n+1}$  is immobilized by regarding  $P_{n+1}$  and  $R_{n+1}$  as  $R_n$  of  $C_n$  because  $P_{n+1}$  and  $R_{n+1}$  are fixed together in space in this case;
- In addition,  $P_{n+1}$  and  $R_{n+1}$  are also immobilized when  $P_n$  is fixed because the local configuration  $[P_n, P_{n+1}, R_{n+1}]$  is interlocking by our requirement.

3) All subsets of  $C_{n+1}$  (except  $P_1$  and  $C_{n+1} - P_1$ , by Lemma 2) are immobilized. Here we divide these subsets into the following six mutually-exclusive cases, where the last four cases account for those subsets with either  $P_{n+1}$  or  $R_{n+1}$ :

- Case 1: Subsets without  $P_{n+1}$  and  $R_{n+1}$ . Since  $C_n$  is interlocking, all these subsets except  $P_1$  are immobilized because  $P_{n+1}$  and  $R_{n+1}$  are fixed in space, behaving like  $R_n$  of  $C_n$ .

- Case 2: Subsets with both  $P_{n+1}$  and  $R_{n+1}$ . Since both  $P_{n+1}$  and  $R_{n+1}$  are included, they can be regarded as  $R_n$  of  $C_n$  when moving together. Again, since  $C_n$  is interlocking, all subsets in this case are immobilized except  $C_{n+1} - P_1$ .
- Case 3: Subsets with  $P_n$  and  $P_{n+1}$  but not  $R_{n+1}$ . By the requirement on  $P_i$ , we cannot move  $P_n$  and  $P_{n+1}$  together relative to  $R_{n+1}$  in configuration  $[P_n, P_{n+1}, R_{n+1}]$ . By Lemma 1, subsets containing them but not  $R_{n+1}$  are immobilized.
- Case 4: Subsets with  $P_n$  and  $R_{n+1}$  but not  $P_{n+1}$ . By the requirement on  $P_i$ ,  $P_{n+1}$  is immobilized in configuration  $[P_n, P_{n+1}, R_{n+1}]$ . By Lemma 2,  $P_n$  and  $R_{n+1}$  cannot move together relative to  $P_{n+1}$  in the same configuration. Again by Lemma 1, we can conclude this case.
- Case 5: Subsets with  $P_{n+1}$  but not  $(P_n$  and  $R_{n+1})$ . Similar to case 4 (by Lemma 2) [Q.E.D.].
- Case 6: Subsets with  $R_{n+1}$  but not  $(P_n$  and  $P_{n+1})$ . Similar to case 3 (by Lemma 2) [Q.E.D.].

Since the above six cases cover all subsets of  $C_{n+1}$ , all subsets of  $C_{n+1}$  except  $P_1$  and  $C_{n+1} - P_1$  are immobilized. Summarizing the above three conditions (1-3), we show that  $C_{n+1}$  is interlocking if  $C_n$  is interlocking.

(iii)  **$C_{n+1}$  is recursive interlocking.** Now, we have to show that if  $C_n$  is recursive interlocking,  $C_{n+1}$  is also recursive interlocking, i.e., all postfix subsequences in  $C_{n+1}$  are interlocking.

Since  $C_n$  is recursive interlocking, all postfix subsequences in  $C_n$ ,  $[P_i, \dots, P_n, R_n]$  (denoted by  $D_i$ ), are interlocking for  $i=1 \dots n-1$ . Since  $D_i$  is interlocking, if we decompose  $R_n$  of  $D_i$  into  $P_{n+1}$  and  $R_{n+1}$ , the resulting configuration, denoted by  $E_i = [P_i, \dots, P_n, P_{n+1}, R_{n+1}]$ , is still interlocking (by our proof for statement (ii) above). Hence, postfix subsequences in  $C_{n+1}$  ( $E_i$ ) for  $i=1 \dots n-1$  are interlocking. Lastly, since  $[P_n, P_{n+1}, R_{n+1}]$ , which is  $E_n$ , is trivially interlocking by our requirement, we can conclude that  $C_{n+1}$  is recursive interlocking.

In our proof for statements (i), (ii), and (iii) above, we show that if  $C_n$  can be assembled, interlocking, and recursive interlocking,  $C_{n+1}$  can also be assembled, interlocking, and recursive interlocking. Hence, by the principle of induction,  $C_n$  can always be assembled, interlocking, and recursive interlocking for any  $n \geq 2$ , and so, puzzle pieces constructed by our formal model (with local interlocking) are directly guaranteed to be globally (recursive) interlocking, regardless of the number of puzzle pieces involved.