Title:
Component-based Building Instructions for Block Assembly

Authors:
Man Zhang, eelzhang@npal.cs.tsukuba.ac.jp, University of Tsukuba
Yuki Igarashi, yukim@acm.org, Meiji University
Yoshihiro Kanamori, kanamori@cs.tsukuba.ac.jp, University of Tsukuba
Jun Mitani, mitani@cs.tsukuba.ac.jp, University of Tsukuba

Keywords:
LEGO, block assembly, building instructions, segmentation

DOI: 10.14733/cadconfP.2016.55-59

Introduction:
A well-designed set of building instructions is crucial because a LEGO sculpture with fragile constructions of blocks might easily fall to pieces during assembly. To avoid fragmentation during assembly, a smart strategy originating from assembling articulated objects [1-3] is to segment a model into solid components, assemble each of them separately, and finally combine them together. However, most block models do not have apparent articulations. For user-friendly assembly, a block model should be divided at weakly-connected blocks, and segmented into as few and as large components as possible to avoid over-segmentation. Also, the preferred assembly orders among LEGO fans seem to be "layer-by-layer and from bottom to top" [5], as these are natural orders for building architecture. However, if building instructions are not carefully designed, as shown in Fig. 1, some blocks might have neither upward nor downward connections during assembly. Such physically-impossible blocks, defined as *floating blocks*, are not rare in instructions generated by existing LEGO design systems [5-7].

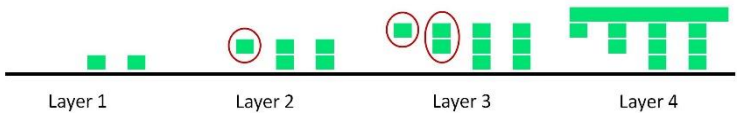


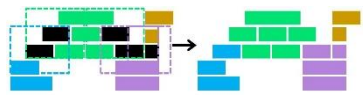
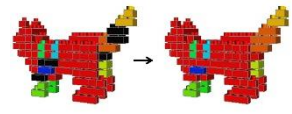
Fig. 1: Blocks floating in the air (red circles) in naive layer-by-layer building.

In line with the principles stated above, we propose a method for automatically generating building instructions for block models. Our method initially generates components, by masterly segmenting a model at the weakly-connected blocks and at the incoherent spots identified by *floating blocks*. During this step, it is ensured that no *floating block* exists in each component. For obtained components, our method further generates a set of building instructions by deciding the assembly order of the components based on our criteria for easy assembly. The effectiveness of our method is demonstrated through a quantitative comparison with other tools as well as a user study that proves users can assemble block models more efficiently using our instructions.

Proposed Method:
The input of our method is an assembled shape of a block model, which can be easily obtained using existing LEGO design software. All blocks are assumed to be rectangular solids having the same height, similarly to the previous methods, e.g., [7]. The output of our method is a step-by-step set of 3D instructions which can be viewed from any angle. For making our building instructions (Subsection 4),

components should be generated from the input in advance. To generate components, we initially segment the input model by independently implementing two types of segmentation (Subsection 1 and 2), and then selectively merge unnecessarily-small segments to make components (Subsection 3).

1. Segmentation at Weakly-Connected Blocks
We detect weakly-connected blocks as blocks corresponding to the previously defined "weak articulation points" [7]. Definition of "weak articulation point" [7] decides that, by removing each weakly-connected block, the model can be separated into multiple disconnected parts, with each part containing more than one block. Inspired by this property, in our segmentation (see algorithm in Tab. 1), all the weakly-connected blocks detected in Step 1 are removed from the initial model in Step 2. Then in Step 3, each weakly-connected block is merged into a segment that has the largest number of connections to the block.

	Operation	*2D Illustration*	*3D Illustration*
Step 1	Detect weakly-connected blocks W in input model M.		
Step 2	Divide M into several segments by subtracting W from M.		
Step 3	Merge each block $w \in W$ into a segment that has the largest number of connections to w.		

Tab. 1: Algorithm and illustrations for segmentation at weakly-connected blocks (black blocks).

2. Segmentation Avoiding Floating Blocks
We first extract the blocks that will be in the floating state during a layer-by-layer, bottom-up assembly. Such *floating blocks* are easily detected as follows. We visit connected blocks from each bottommost block to the top. The allowed visiting direction is only upward, because in a LEGO model two blocks are directly connected to each other only if they overlap each other. The blocks that have not been visited by the end are *floating blocks*. Fig. 2(a) illustrates a simple LEGO model in 2D with *floating blocks* (colored in red).

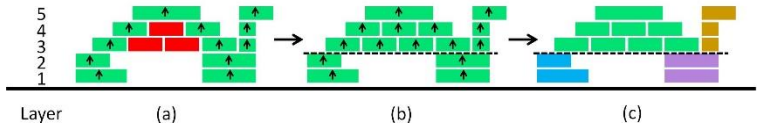
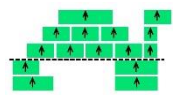
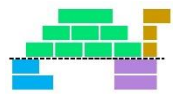
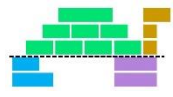


Fig. 2: Segmentation avoiding *floating blocks*. (a) Detecting *floating blocks* (red blocks) along arrows. (b) A *pseudo floor* is inserted between the 2nd and 3rd layers. (c) The model is divided into four segments.

Our final target is to ensure that there are no *floating blocks* in each component generated. Here, we introduce two strategies to achieve this goal. One is a direct way, the other indirect. The direct way is to explicitly separate the *floating blocks* as components. Note that, in Fig. 2(a), if we treat the *floating blocks* (colored in red) and the rest (colored in green) as two different components, each component contains no *floating blocks* inside. On the contrary, the indirect way is separating the model by horizontal planes until no *floating block* exists as illustrated in Fig. 2(b). The dashed line shows the horizontal plane used to separate the model into components. This separation works as if we had inserted a working floor between the 2nd and 3rd layers. We call this separating plane a *pseudo floor*. Generally, it is difficult to use only one *pseudo floor* to eliminate all the *floating blocks*. In Fig. 2(c), due to the separation by one

pseudo floor, each independent component obtained is occasionally able to be assembled from the bottom to the top without any *floating block*.

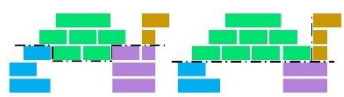
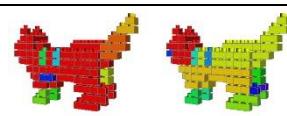
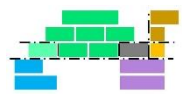
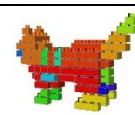
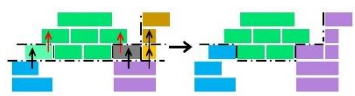
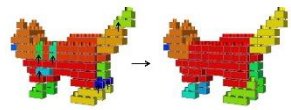
Now we have two strategies to generate components with no *floating blocks* inside. We further combine both strategies to reduce the amount of segments, which will benefit a more precise instruction. As shown in Tab. 2, our algorithm do so by applying indirect way first, however, not for eliminating all *floating blocks*, but for reducing *floating blocks* reasonably by inserting a few effective *pseudo floors*. After that, we use the direct way to handle the unreduced *floating blocks*.

	Operation	*2D Illustration*	*3D Illustration*
Step 1	For each l, calculate a $C_{floating}(l)$ by pre-inserting one horizontal *pseudo floor* between the l-th and (l+1)-th layers. Then insert *pseudo floors* where $C_{floating}(l)$ takes on local-minima.		
Step 2	Indirect segmentation: Divide the input model by the inserted *pseudo floors*.		
Step 3	Direct segmentation: detect *floating blocks,* and then explicitly separate them as new segments.		

Tab. 2: Algorithm and illustrations for segmentation avoiding *floating blocks*. In Step 1, $C_{floating}(l)$ equals the number of *floating blocks* had by inserting a *pseudo layer* between the l-th and (l+1)-th layers for l = 0, 1, 2, ... (l = 0 means the ground floor). For 3D illustrations, in Step 1 *floating blocks* (red blocks) exist when the model is separated by two *pseudo floors*. For 2D illustrations, in Step 1 no *floating block* exists after inserting a *pseudo floor*, therefore no difference exists between Step 2 and 3.

3. Making Components

Our component generation flow is described in Tab. 3. Note that components are initialized by dividing the model along the boundaries of segments generated using the two approaches mentioned above. This generates tiny components. Therefore in Step 3, we merge them in four steps to reduce the number of components. i) Find a component "A" which touches a *pseudo floor*, or contains only one or two blocks; ii) Find a component "B" which connects component "A"; iii) Merge component "A" and "B" only if the merged component does not generate additional *floating blocks*; iv) Repeat i) to iii) until all possible merge operations are done.

	Operation	*2D Illustration*	*3D Illustration*
Step 1	Segment by two algorithms: driven by weakly-connected blocks (left), and avoiding *floating blocks* (right).		
Step 2	Generate components by dividing the model along the boundaries in both segmentations generated by Step 1.		
Step 3	Merge components as much as possible, ensuring no *floating block* is generated.		

Tab. 3: Algorithm and illustrations for making components from segments. In Step 3, black arrows upward indicate successful merging. Failed merging cases are indicated by red arrows in 2D illustration.

4. Making a Component-Driven Instruction

By now it is ensured that no *floating blocks* exist in any component. Hence, we can simply assemble each component in bottom-to-top order, and focus only on the order of combining components. Among the components, we define a *joint component* as one connecting two or more other components. The assembly order of components is decided according to the following priorities: i) the number of connected components; ii) the number of blocks contained in the component; iii) the number of connected *joint components*; iv) the distance from the bottommost block (smaller has higher priority). If value i) is the same for each component, value ii) is used to decide the priority. Furthermore, if value ii) is the same for each component, value iii) is used, and so on. Finally, if symmetrical components-pairs exist, the order of components is further adjusted to ensure successive assembly of such symmetric component-pairs. After deciding the assembly order of all the components, we generate a graphical instruction guide as shown in Fig. 3.

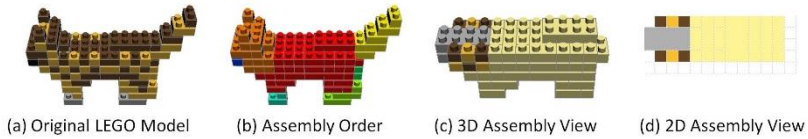


Fig. 3: Our graphical instruction guide. The guide firstly switches from (a) to (b) for showing assembly order of components (red first, blue last), and then enters the assembly of each component layer-by-layer. Blocks in current layer are displayed in two views (c, d). We render the original block color only for the component being assembled, leaving others rendered in a customized color (e.g., beige).

Results:

We developed a prototype system to evaluate our method. It was implemented using C++ and tested on a laptop with a 2.40-GHz Intel Core (TM) i5-2430M processor, 8 GB RAM, and NVIDIA NVS 4200M GPU. We prepared seven block models (Fig. 4) by using a mini block artwork design system [8].

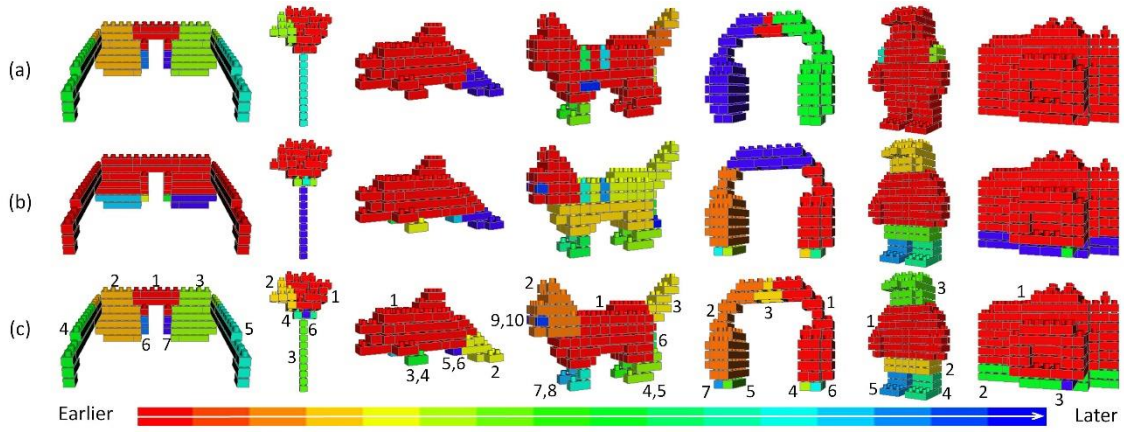


Fig. 4: Generation and ordering of components in different models. (a) Segments separated at weakly-connected blocks. (b) Segments avoiding *floating blocks*. (c) Final components generated. Assembly order of final components, as marked by numbers, is associated with a specific color in a color map varying from red (built first) to blue (built last).

Generation and Ordering of Components

Segmentation in our method is driven by weakly-connected blocks and *pseudo floors* found in input model. Although both segmentation steps result in unnecessarily tiny components (Fig. 4(a, b)), our merging strategy successfully combines tiny components into large ones for better results (Fig. 4(c)).

Assembly order of components determined by our method is illustrated in Fig. 4(c) as well. As expected, *joint components* are to be built earlier (in a warmer color), and symmetric component-pairs to be built successively are in similar colors. This demonstrates the effectiveness of our ordering.

Auto-generation of Instruction Guide

Tab. 4 compares the instruction guide generated by our method with those generated by LDD (LEGO Digital Designer [4]) and LIC (LEGO Instruction Creator [5]). We used in our test the cat model shown in Fig. 4, which consists of 93 blocks and 7 weakly-connected blocks in total. Between the two layer-by-layer guides, we found that the guide generated by LIC showed some steps with *floating blocks*, while our guide avoided *floating blocks* inside each component. To compare with the block-by-block guide, we recruited four undergraduate volunteers to test the time efficiency of the instruction guides generated by our system and LDD. The results showed that all subjects completed the cat model in much less time when using our instructions. The average time needed to complete the model with our instructions was 17 min, which is about 60% of the time needed with LDD's instructions.

	# of components	*Max/Min # of blocks in component*	*Instructions steps for component*	*With first block for*
Our system	10 (see Fig. 5)	53/1	layer-by-layer	body
LEGO Digital Designer	2 (body & tail)	89/4	block-by-block	foot
LEGO Instruction Creator	1 (whole)	93	layer-by-layer	foot

Tab. 4: Step-by-step instructions created by three systems.

Conclusions and Future Work:

To help the efficient assembly of fragile LEGO models, we proposed a method for automatic generation of component-based building instructions. Components are obtained considering segmentation at both the weakly-connected blocks and the incoherent spots identified by *floating blocks*. We implemented our method and evaluated the efficiency of the generated instructions.

Our method can also evolve to satisfy more requirements, e.g., the static equilibrium during assembly. Moreover, diverse notations shown in manually drawn instructions might be considered to make the automatically generated instructions more user-friendly.

References:

[1] Agrawala, M.; Li, W.; Berthouzoz, F.: Design principles for visual communication, Commun. ACM, 54(4), 2011, 60-69. http://dx.doi.org/10.1145/1924421.1924439

[2] Agrawala, M.; Phan, D.; Heiser, J.; Haymaker, J.; Klingner, J.; Hanrahan, P.; Tversky, B.: Designing effective step-by-step assembly instructions, ACM Transactions on Graphics (Proc. SIGGRAPH 2003), 22(3), 2003, 828-837. http://dx.doi.org/10.1145/882262.882352

[3] Heiser, J.; Phan, D.; Agrawala, M.; Tversky, B.; Hanrahan, P.: Identification and validation of cognitive design principles for automated generation of assembly instructions, Proc. AVI'04, Gallipoli (Lecce), ITALY, 2004, 311-319. http://dx.doi.org/10.1145/989863.989917

[4] Lego digital designer, http://ldd.lego.com, LEGO.

[5] Lego instruction creator, http://bugeyedmonkeys.com/lic_info, 2010 Remi Gagne.

[6] Luo, S.-J.; Yue, Y.-H.; Huang, C.-K.; Chung, Y.-H.; Imai, S.; Nishita, T.; Chen, B.-Y.: Legolization: Optimizing LEGO Designs, ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2015), 34(6), 2015, 222:1-12. http://dx.doi.org/10.1145/2816795.2818091

[7] Testuz, R.; Schwartzburg, Y.; Pauly, M.: Automatic generation of constructable brick sculptures, Proc. Eurographics 2013, Girona, SPAIN, 2013, 81-84. http://dx.doi.org/10.2312/conf/EG2013/short/081-084

[8] Zhang, M.; Igarashi, Y.; Kanamori, Y.; Mitani, J.: Designing mini block artwork from colored mesh, Proc. Smart Graphics 2015, Chengdu, CHINA, 2, 2015, 1-12.