Automatic Generation of Constructable Brick Sculptures

Romain Testuz, Yuliy Schwartzburg and Mark Pauly

École Polytechnique Fédérale de Lausanne, Switzerland

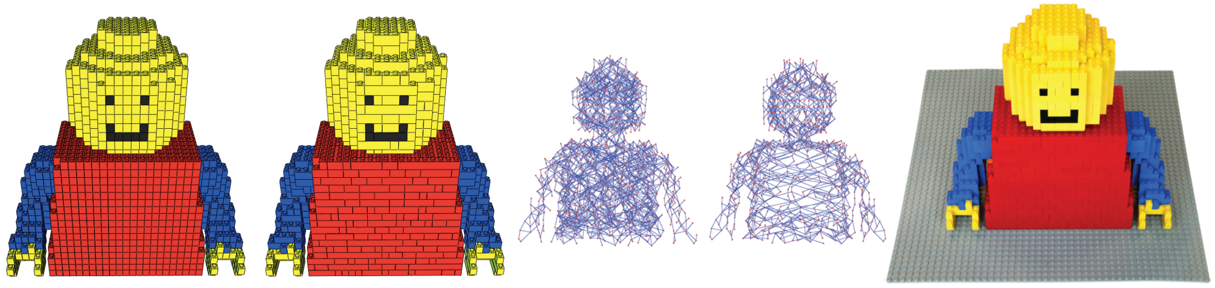


Figure 1: A demonstration of our method from start to finish. The LEGO Man is first voxelized into 1x1 bricks and the bricks are merged respecting color. Then, the bricks are optimized for structure and extraneous bricks are removed. Finally, instructions are produced and a LEGO model is built from the instructions.

Abstract

Fabrication of LEGO® models in large scale requires careful pre-planning to produce constructable and stable models. We propose a system that, starting with a voxelization of a 3D mesh, merges voxels to form larger bricks, and then analyzes and repairs structural problems, finally outputting a set of building instructions. We also present extensions such as producing hollow models, fulfilling limits on the number of bricks of each size, and including colors. Results (both real and virtual) and timings show significant improvements over previous work.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

Introduction

The generation and rationalization of 3D models for fabrication has recently become a topic of great interest in the computer graphics community. However, most fabrication methods still require the user to have expensive equipment such as laser cutters or 3D printers. LEGO®, a popular toy construction system, is comparatively cheap and nearly ubiquitous. However, building arbitrary 3D models out of LEGO manually often involves significant trial-and-error. This process requires approximating a 3D model out of a limited set of pieces and ensuring the sculpture to be connected, stable and constructable. The goal of this paper is to automatically create the set of instructions for a LEGO model from a 3D object representation. By doing so, we highly simplify the task of building a large customized LEGO model.

The LEGO Group, the company that produces LEGO toys, has twice openly presented this problem to the scientific community–first in 1998 and later in 2001 [Pet01]. Our approach is inspired by previous work [vZS08] in the sense that the algorithm starts from the voxelization of a model into the smallest possible bricks, namely the 1x1 bricks (see Figure 2) and merges those to form larger bricks. However, merging bricks greedily makes it highly unlikely that the model is buildable or stable. To resolve this, we propose a graph-based algorithm to ensure brick connections and resolve structural weaknesses. Furthermore, our method can reduce the brick number by hollowing the model, allows the user the specify limits on how many bricks of a certain type can be used, and can even take into account the coloring of bricks.

1. Related Work

After the LEGO Group proposed this problem, a first attempt was done by a group of mathematicians in a one week workshop [GHP98]. They created a cost function for a simulated annealing technique but they did not implement or test it. The next attempt used evolutionary algorithms with a cost function inspired from [GHP98]. The results describe the

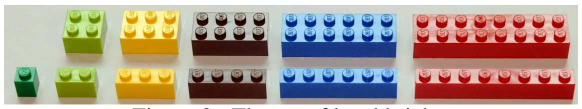


Figure 2: The set of legal bricks.

performance of the evolutionary algorithm itself (number of generations before convergence, etc...) but they do not report whether this method can actually make a LEGO model that is constructable [Pet01]. Moreover, the required time that they report for optimization is 5 to 11 hours. Another attempt was done using the beam search technique but there is no data about experimental results [Win05]. Van Zijl and Smal compare existing approaches and propose another approach based on cellular automata [vZS08]. They use a similar merge/split formulation but use several heuristics rather than a graph connectivity formulation. The reported results have a long optimization time of about a few minutes and there is no information on the solidity of the model. All the above methods are based on the cost function proposed by [GHP98]. [SPC09] looks into how to transform a mesh into a LEGO representation but this is only meant for realistic 3D rendering and does not aim at making the model buildable.

2. General pipeline

As in previous methods, we first represent the object as 1x1 LEGO bricks. We transform a mesh representation of an object into a discrete set of voxels using the method of [NT03]. The choice of voxel resolution is important as a higher value results in a better approximation of the original object, but would require more bricks and more time to build.

After conversion, we sequentially merge the 1x1 bricks in a greedy fashion. Given a legal set of bricks, we make them as large as possible until no further merges can be done (see Figure 2). At this point, the model is very likely to be weakly connected and possibly disconnected leading to an unbuildable structure. Therefore, we increase the solidity of the model by identifying the weaknesses and repairing them. Finally, the user can save the instructions as images representing the brick layout of each layer to facilitate building, or a video of the building process can be generated.

2.1. Merge algorithm

In order to decrease brick count and increase connectivity, we prefer larger bricks. We can do this simply by merging bricks with their neighbors. We use a randomized greedy merge algorithm as follows:

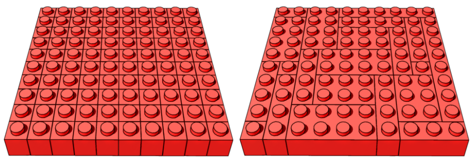


Figure 3: The initial merge step.

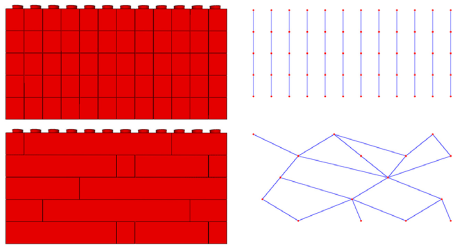


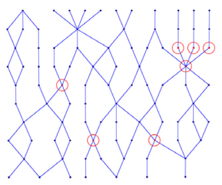
Figure 4: Two brick layouts (left), and their respective graph representations (right).

1. Choose a brick in the model at random.
2. Find the legal set of neighbors with which the brick can be merged.
3. Select the neighbor with the lowest cost value and merge.
4. Goto step 2 until there are no more mergeable neighbors.
5. Goto step 1 until no brick can merge.

Note that for step 2 only a specific set of LEGO bricks is considered (the "legal bricks"). For our examples, we use the set shown in Figure 2, but arbitrary other legal sets can be specified by the user. For step 3, we favor the brick which when merged with the current brick, will create the most connections (two bricks are connected if they are on adjacent levels and they have at least one knob overlapping). If two merges create the same number of connections, we choose between them randomly. This allows us to save optimization steps in the next section. The result of this algorithm for a 10x10 grid can be seen on Figure 3.

2.2. Solidity Optimization

The stability of the construction is related to how the bricks are connected: the more the bricks of a model are connected to each other, the stronger it will be. This observation motivates the mapping of our LEGO brick representation to a graph representation where each brick represents a vertex and each connection between two bricks represents an edge. In Figure 4, we illustrate the equivalence between a toy brick layout example and its associated graph; see Figure 1 for a more complex example. With this representation, we can analyze the connectivity of the LEGO model to determine weak points.



The number of connected components in the graph directly relates to solidity as pieces can simply fall off if they are not connected to the rest. Two (non-trivial) subgraphs that are only connected to each other by one brick also weaken the structure. We call the brick connecting subgraphs of size greater than 1 a *weak articulation point*.

Using these measures, we change the brick layout to increase solidity. After Section 2.1, the bricks are at their maximum extent. Therefore, we split each of the bricks at the in-

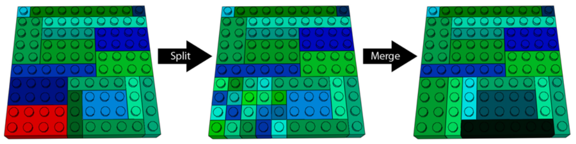
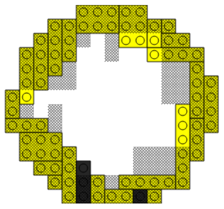


Figure 5: The process of removing a weak articulation point.

terface between two connected components or neighboring an articulation point into 1x1 bricks. Then, we simply run the merge algorithm again, changing only the cost function to a random cost function. We perform this process iteratively until the number of connected components and weak articulation points no longer decreases. In Figure 5, we can see the split process for an weak articulation point (in red), note that only the layer containing the weak articulation point is displayed for better visibility.

We tested with a dozen models at various scales, and we find that we need under 50 iterations to have no disconnected components and no weak articulation points for most models. Unfortunately, we cannot know beforehand if the algorithm will converge to the ideal case. There may be thin regions where articulation points cannot be removed (such as the ears of the bunny at very coarse voxelizations), or specific voxelizations that result in disconnected components. In these cases, the input mesh would need to be changed to result in a completely stable structure. Nevertheless, if the user does not need a specific voxel resolution, they can iterate with increasing voxel count until a completely stable model is reached. See Table 1 and the additional materials for experimental results.

2.3. Assembly Instructions



After the previous steps are completed, the user can save the assembly instructions in order to build the model. These correspond to the layout of each layer. To help the user align a new layer above a previous one, we show the layer below shadowed. The inset shows the assembly instructions for layer 21 of the LEGO Man. The two black bricks correspond to the eyes.

3. Extensions

Besides the basic pipeline, we introduce several extensions to facilitate the building of the model. In order to reduce the brick number and computation time, the model can be pre-hollowed right after voxelization. After running the pipeline, once the model has no more structural weaknesses, several other steps can be performed to facilitate building of the model. The model can again be hollowed to remove unnecessary bricks, and bricks of certain types can be removed to fulfil a specified quantity of each brick. We can also introduce color into the process.

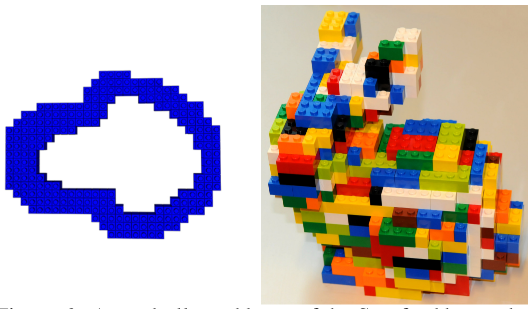


Figure 6: A pre-hollowed layer of the Stanford bunny before optimization (left) and the final built bunny (right).

Reducing the overall brick number Reducing the quantity of bricks can allow for easier and cheaper construction without significantly harming the stability of the construction. If the model is filled with bricks it will require much more bricks than if it is hollow. We have therefore devised two strategies for reducing the brick count by hollowing the model.

Pre-hollowing. Before the pipeline, the user can specify a shell size, and we remove the voxels which are further than than that number of bricks away from the outside of the figure in any direction (not just in that plane). One layer of a hollowed Stanford bunny with shell size 2 is shown in Figure 6. An advantage of pre-hollowing is a faster optimization process during the pipeline. The user can change the shell size but usually 2 is sufficient for stability even for complex models.

Post-hollowing. Pre-hollowing is fast but does not result in a minimal number of bricks. Another technique to reduce brick count is to remove inside bricks without compromising the model solidity. We remove pieces without introducing more connected components or more weak articulation points. For each brick in the inside of the model we consider a subgraph of the connectivity graph centered at the brick. We then remove the brick if its removal does not add any weak points. This method can sometimes result in pillars of bricks that form a path through the middle of the model. Therefore, we can combine pre-hollowing and post-hollowing to start with a shell and then remove any extraneous bricks.

Satisfying brick type limits When building a model with a set kit of bricks, there are set limits to each type of brick. For example, there can be many more 1x2 bricks and not enough 1x4 bricks. Limiting the brick type greedily during merging often does not result in a solid model. We therefore use an approach similar to post-hollowing. As a post-process, we remove bricks over the limit by cutting them into two (legal) smaller bricks. As in post-hollowing, we choose the split as to not to increase the number of connected components or weak articulation points. If every split causes weak points, we go on to the next brick.

		no post-hollowing		with post-hollowing			
Mesh	Voxels	Time(sec)	Brick #	Time(sec)	Brick #	Conn Comp	Weak Pt
EROS	15144	4.66 ± 1.35	3820 ± 25.5	5.51 ± 1.40	3110 ± 27.8	1 ± 0	0.100 ± 0.300
BUNNY	11472	26.5 ± 5.98	2900 ± 21.4	27.0 ± 6.00	2380 ± 29.5	1 ± 0	7.20 ± 2.48
FERTILITY	6859	2.46 ± 1.21	1610 ± 17.9	2.78 ± 1.20	1400 ± 19.5	1 ± 0	0.050 ± 0.218
KITTEN	12887	2.04 ± 0.728	3340 ± 28.4	2.72 ± 0.705	2650 ± 27.1	1 ± 0	0 ± 0
LEGOMAN	9961	2.17 ± 0.846	2120 ± 24.9	2.50 ± 0.839	1800 ± 21.3	1 ± 0	0 ± 0

Table 1: Mean values and standard deviations for 20 trial runs of 5 models at 50 layer resolution. Note that the randomized algorithm produces consistent results across different trials.

Using colors We can also allow for different colors of LEGO bricks. We initialize the colors for each brick by finding the color of the original mesh texture on the point closest to the center of the brick. We then round this color to the closest LEGO brick color. Then, during step 2 of the merge algorithm (see Section 2.1), another verification is added to allow merging of two bricks: if both bricks are outer (visible) bricks and they have different colors, then they cannot be merged. Inner bricks can be merged regardless.

4. Results

Table 1 summarizes timings and final brick counts for different models pre-hollowed with a shell size of 2 (measurements were done using a 1.8 GHz processor). The time is measured from the start of the first merge to the final result which consists of a single connected component and no weak articulation points. The amount of bricks removed by post-hollowing in each case is close to 20% of the number of bricks before the operation.

It is difficult to compare the results with those of [vZS08] since they do not have a solidity measure. For example, if we compare the results for the cube with 32 layers, we know that our cube is solid but we cannot say the same for theirs. They report a time of 197 seconds and a brick count of 2,128. If we suppose that both are solid than our algorithm is orders of magnitude faster while using only slightly more bricks.

Using the instructions produced by our method, we built a 17 layer hollow Stanford bunny with 314 bricks (see Figure 6). We also built a bust of a LEGO figurine (see Figure 1) which takes color and brick type limits into account, consisting of 30 layers and using 1,315 bricks.

5. Future work

The voxelization algorithm may sometimes result in aliasing artifacts. Furthermore, we use the texture of the closest point on the mesh to determine the color of the brick, which can also cause aliasing. These can be replaced with a more sophisticated scheme based on feature detection or by a user-assisted painting and voxel insertion and deletion UI.

The current layer-by-layer instructions have the drawback of making it difficult to add bricks only supported by the layer above. Another way of displaying the instructions would be an interactive visualizer which displays each steps of the assembly similar to the Autodesk Inventor Publisher Mobile Viewer or the LEGO Digital Designer applications.

By taking into account the weight of each pieces and the gravity, it should be possible to check that the model is able to stand without falling. If the center of mass is not properly place, it could be moved by adding pieces in the inside of the model. The definition of weak articulation points can be expanded to those that support a load over a certain threshold. Finally, the code could be parallelized to take advantage of multi-core processors.

6. Conclusion

In this paper, we showed that it is possible to automatically create assembly instructions to build a mesh from LEGO bricks using simple graph-based algorithms. Our method is faster and more accurate than the existing approaches and has the advantage of reporting whether the model is solid or if it has some weaknesses. We can also account for brick type limits and colors, and can produce a LEGO sculpture with a minimal number of bricks.

References

[GHP98] GOWER R., HEYDTMANN A., PETERSEN H.: *LEGO: Automated Model Construction*. Jens Gravesen and Poul Hjorth, 1998, pp. 81–94. 1, 2

[NT03] NOORUDDIN F., TURK G.: Simplification and repair of polygonal models using volumetric techniques. *Visualization and Computer Graphics, IEEE Transactions on 9*, 2 (2003), 191–205. 2

[Pet01] PETROVIC P.: *Solving the LEGO brick layout problem using evolutionary algorithms*. Tech. rep., Norwegian University of Science and Technology, 2001. 1, 2

[SPC09] SILVA L., PAMPLONA V., COMBA J.: Legolizer: A real-time system for modeling and rendering LEGO representations of boundary models. In *Computer Graphics and Image Processing (SIBGRAPI)* (2009). 2

[vZS08] VAN ZIJL L., SMAL E.: Cellular automata with cell clustering. *Automata-2008: Theory and Applications of Cellular Automata* (2008), 425. 1, 2, 4

[Win05] WINKLER D.: Automated brick layout. *BrickFest* (2005). 2