A Lego Rewriting System for Fabricating 3D Models

Manfred Lau

City University of Hong Kong, Hong Kong; manfred.lau@gmail.com

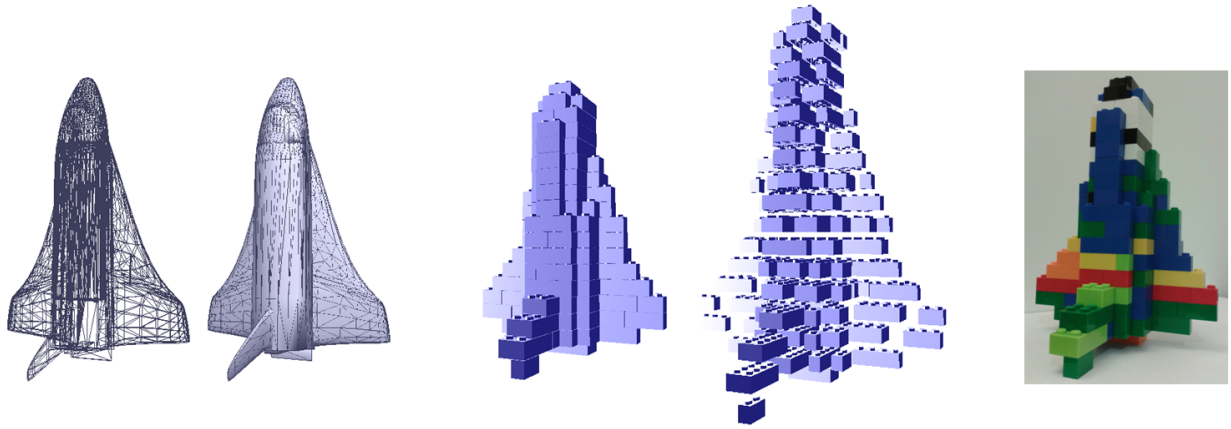

Figure 1: *Left two images: Input 3D model of space shuttle (m1398) from the Princeton Shape Benchmark. Middle two images: Our method finds an arrangement of Lego bricks (connected arrangement and exploded view) that resemble the input model. Right: We fabricate a corresponding physical object with real Lego bricks (111 bricks, and color of bricks are randomly chosen). Please see the video (https://tinyurl.com/47n7t4ub) for more results.*

Abstract

Although there exists large real-world shapes and structures made from Lego, the arrangements of the Lego pieces in these shapes are often designed manually. In this paper, we solve the problem of taking a 3D model as input, and generating an arrangement of individual Lego pieces that can be assembled together to form a physical object resembling the input model. We define a novel Lego rewriting system that is both grammar-based and data-based. Our Lego rewriting system describes how Lego pieces can be connected to each other to form larger shapes, and we use this system to analyze 3D models to solve our problem. We demonstrate our approach with 3D models from the Princeton Shape Benchmark and Trimble 3D Warehouse.

1 Introduction

There exists many real-world shapes and structures made from Lego. For example, the Lego Architecture series consists of small versions of buildings and landmarks built from Lego pieces. On a larger scale, Miniland USA in Legoland California uses about 20 million Lego pieces to reproduce 1-to-20 scaled versions of famous buildings and regions worldwide. However, the arrangement of the Lego pieces for fabricating these structures are often designed manually. The Architecture series, for example, has an architectural artist for designing the pieces for each landmark. We solve the problem of taking a 3D model as input, and generating an arrangement of individual Lego pieces that can be connected together into a shape that resembles the original model. The intention is to build scaled versions of the original models for prototyping and/or artistic purposes. A simple greedy method of just "filling-in" the 3D space with Lego pieces is not adequate since there are many challenges to this problem, including: finding a valid (i.e. connectable and ordered sequence) combination of pieces, generating a stable arrangement of pieces, handling a large number of pieces and different types of pieces, and generating a physical shape that resembles the input model as closely as possible.

The visualization work of Agrawala et al. [1] and Li et al. [16] are related to our work. While we solve the problem of starting with a 3D model and generating individual Lego pieces as output, they solve the problems of generating step-by-step assembly instructions [1] and exploded view diagrams [16] by starting with the individual parts (which can be Lego pieces) as input.

We focus on the fundamental Lego pieces [3]: Lego bricks and slopes. We take a grammar-based framework, which is a general representation that could be used for other purposes besides our Lego fabrication problem such as shape analysis and shape editing. More specifically, we define a novel Lego rewriting system describing how Lego pieces can be assembled together to form larger shapes. The system "rewrites" Lego pieces as if rewriting strings in rewriting systems that define languages. Our system is a Semi-Thue system with an alphabet consisting of various types of Lego pieces (such as 1x2 bricks and 2x4 bricks), and rewriting rules that describe valid ways of assembling the pieces together. The rewriting system describes the set of 3D shapes that can be built from the corresponding pieces and rules, such that a shape that can be generated from the system is valid in that all pieces are connected together as one shape.

Our framework is also data-based, such that it has the ability to indirectly and automatically encode human preferences of the configurations (i.e. how humans are likely to put the pieces together), in addition to information about the local stability and aesthetics of the object being built. Our rewriting rules consist of information for each rule describing how often it is used in example data. This data is collected beforehand by observing how humans build various shapes with Lego pieces. While it is known that certain methods such as staggering [3] are preferred for stability reasons, in general the "concept of good brick placement is something that the LEGO builders and children who play with LEGO learn from experience" [10]. Our framework encodes such concepts of good local brick placement in the rewriting rules.

Our algorithm uses the Lego rewriting system to analyze the input 3D model to solve our problem. We first convert the 3D mesh model into its voxel representation. Structural analysis then uses the rewriting rules and a backward search to give an ordered sequence of Lego pieces for assembling a connected shape. This works well for building small shapes. However, there are endless combinations of assembling even a small number of types of Lego pieces. While this is the fun and appealing aspect of Lego, the large number of combinations does not allow for an efficient analysis. For building large shapes, we avoid the combinatorial explosion of states in the search by using example data (encoded in the rules) to reduce the branching factor and a hierarchical approach to reduce the depth factor.

We demonstrate our approach with a variety of 3D models available online (Princeton Shape Benchmark [31] and Trimble 3D Warehouse). The input can be any arbitrary 3D mesh. Our contributions are: (1) we define a novel Semi-Thue Lego rewriting system with an alphabet consisting of various types of Lego pieces; (2) our Lego rewriting system is data-based such that it has the ability to indirectly encode human preferences of the configurations in addition to information about local stability and aesthetics; and (3) we solve the problem of fabricating a virtual 3D model with Lego pieces using this rewriting system to guide our analysis.

2 Related Work

3D Modeling with Lego. There are previous works that analyze and/or build physical Lego structures from 3D models. For example, previous works have performed a force-based analysis of the stability of the object being built [18], considered the stability, aesthetics and efficiency of the Lego model in an equation [37], and used centroid adjustment and inner engraving to make a LEGO sculpture that is balanced [12].

Previous works have also solved this Lego problem with various approaches. The problem can be formulated as a combinatorial optimization [10]. A deformation algorithm can be used to resolve discrepancies between an input mesh's continuous 3D shape and the discrete positions of bricks in a LEGO sculpture [38]. A search approach has been used to find brick arrangements that reduce the number of connected components, undesirable edges, and bricks [33]. A graph structure called "legograph" can convert 3D models into LEGO

brick building instructions [25]. Genetic Algorithms have been applied to solve the Lego problem [27, 15]. The idea of combining standard LEGO bricks and custom 3D printed bricks has been explored [19, 9]. One previous work solves a related but slightly different problem for Lego Technic pieces [36]. Some of these works are summarized in a survey [13].

In contrast, our work takes a grammar-based framework, which is a general representation that could be used for other purposes such as shape analysis and shape editing. More specifically, we have a novel semi-Thue Lego rewriting system with an alphabet consisting of types of Lego pieces. Furthermore, our system is data-based, in the sense that the rewriting rules can contain additional data about the frequency of their use by users to assemble Lego structures. This data allows for the ability to indirectly and automatically encode how humans are likely to put the pieces together, in addition to information about the local stability and aesthetics of the pieces when assembled.

Grammar-based Modeling. Previous work has used grammar-based techniques for analyzing and modeling plants [28], cities [26], details of facades [35], buildings [24], and parcels in city blocks [34]. Both the related problem of starting with a small example shape and using procedural modeling to generate larger shapes [21], and its inverse procedural modeling problem [4] have been explored. The key difference in our work is that we define a semi-Thue system for Lego, which is simpler and more appropriate than a formal grammar for our purpose. While a semi-Thue system and a formal grammar are similar in that they both consist of rules, they are different in two ways. First, a formal grammar has terminal and non-terminal symbols. This separation of symbols is not needed in our case and we simply have one type of symbols. Second, a formal grammar is only interested in strings without non-terminal symbols that are generated by some sequence of rules from a special starting symbol. We do not need a special starting symbol (as this is incorporated in our rules) and we consider all the strings (or 3D shapes in our case) that can be generated.

Fabricating Physical Objects from Virtual Models. There has been work on taking 3D digital models and fabricating physical equivalents of them with different materials and for different purposes. Paper, cloth, and wood have been used to fabricate papercraft toys [22], plush toys [23], and chairs [30] and furniture [14]. 3D models can be represented with planar sections and fabricated as slices with a laser cutter [20, 11]. Besides static models, articulated characters and posable models can be fabricated [2, 5]. Physical replicas of digital models can be sculpted with different materials [29]. Large models can be built by partitioning them into smaller 3D-printable parts [17]. Virtual models can also be built using a set of prefabricated universal blocks to assemble the internal core of the shape, and a small number of 3D printed pyramidal pieces for the residual volume [7]. Our work also fabricates physical objects from virtual models. The novelty of our work is in formalizing a general Lego rewriting system, and solving the fabrication problem with this system.

Song et al.'s [32] work is related in that they also use Lego pieces. However, their focus is on creating interlocking 3D puzzles. They used Lego pieces as an example to create their interlocking pieces, but other material such as wood or plastic can also be used. They do not explore the fundamental ability of Lego pieces to connect with each other. In contrast, our focus is on exploring Lego pieces themselves as the fundamental building block and how the pieces may connect with each other.

Mesh Segmentation. There exists various methods for segmenting 3D meshes, and the 3D mesh segmentation benchmark by Chen et al. [6] compares between many of these methods. In our work, segmentation is not the main focus, and we perform a simple segmentation after voxeling the 3D mesh to split it into smaller parts for efficient analysis.

3 Lego Rewriting System

Semi-Thue System for Lego. We formalize the process of Lego building as a semi-Thue system [8]. Our system consists of (Σ, R), where Σ is an alphabet consisting of types of Lego pieces, and R is a set of rewriting rules describing how to form valid 3D shapes from the alphabet. There can be shapes buildable

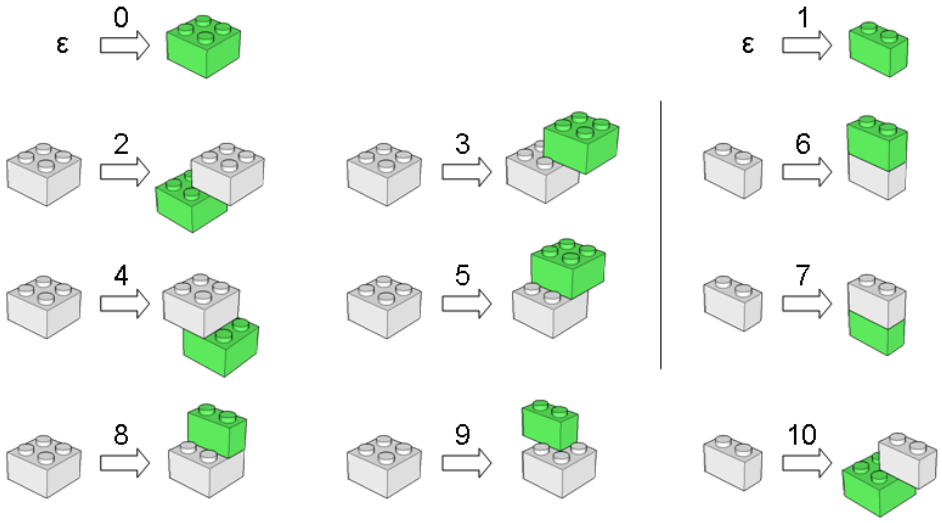

Figure 2: *Example of rewriting system* (Σ, R). *The alphabet* Σ *includes two types of Lego pieces: 1x2 bricks and 2x2 bricks. The set of rewriting rules* R *are shown here.* ε *is the empty shape. Redundant rules that can be formed by rotation are not drawn. The green bricks are newly added in each rule. There needs to be empty space (not explicitly drawn) under or above the newly added piece before a rule can be applied.*

from the pieces in Σ that a particular system (Σ, R) does not describe.

In general, R is a binary relation on the 3D shapes that can be created from the Lego pieces in Σ. Each element of R is a rewriting rule $u \rightarrow v$. Each rewriting rule can be applied if the pattern u matches with the same subpattern within an existing shape, in which case the subpattern is replaced with v. In our case, the rewriting rules directly describe how the pieces can be connected to each other. Our rules are in the form: u is a connected set of piece(s), and v includes those same piece(s) together with more piece(s) that are added to and connected to them. u can be empty (ε) and v is non-empty. We construct our rules in this form such that they can generate valid 3D shapes. We define valid as *connectable* and constructable as an *ordered sequence*. *Connectable* means that we can guarantee that the piece(s) added in each rule can be physically added to the existing shape and hence they are connected to the whole shape. An *ordered sequence* provides one possible sequence of pieces and rules for assembling the final shape. While not necessary, this implies that each of the immediate shapes produced after every rule are connected together as one shape. By having such rules, the 3D shapes that our system (Σ, R) describes can be fabricated with real Lego pieces.

Concrete Example for Lego Bricks. We show an example of a rewriting system for Lego bricks (Figure 2). In this example, each rule $u \rightarrow v$ is either a starting rule or a rule in the basic form. A starting rule has an empty u, and a single piece from Σ in v. A rule in the basic form has a single piece from Σ in u, and the same piece together with another piece added to and connected to it in some position/orientation in v. The rewriting rules are generated manually. It is common in the study of both human and programming languages to create such rules manually for the purpose of analysis. The rules are intuitive to define as they are directly related to how each type of piece can be added to an existing shape.

Even though there are a large number of ways to assemble the pieces, there are some arrangements that are not common or not used at all in practice. On the other hand, certain arrangements such as staggering and overlapping [3] are preferred for stability and aesthetics reasons. We indirectly consider such factors by extending our Lego rewriting system to include example data. Each rewriting rule consists of information describing how often it is used in the example data. As an example, in Figure 2, some rules such as rule 8 has a higher frequency than rule 9, as it is preferred more by humans in example data. It also happens in this case to likely become more locally stable, and it is also likely to be more aesthetic locally. Such human

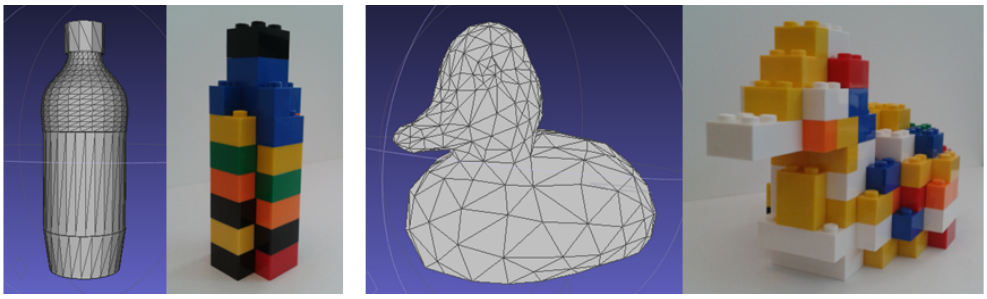

Figure 3: *Collecting data from users: two examples of the virtual shapes (m490 and m48 from the Princeton Shape Benchmark) shown to the users and corresponding real shapes built by them.*

preferences and information about local stability/aesthetics are indirectly encoded in the rules and hence the overall grammar, such that we will prefer to use rule 8, for example, with a higher priority in the backward tracing procedure (Section 4) which leads to better local brick placements.

The example data is collected beforehand by observing how users build various shapes with Lego pieces. The users are given virtual 3D shapes that they can visualize from any viewpoint. The users are asked to build each shape to closely resemble the original at a small scale and with a given set of piece types (but with no fixed limit on the number of pieces). We had two novice users, and one user who has more than ten years of personal experience with Lego. Each user built five different shapes (Figure 3 shows some examples). We record each user session with a video camera, and extract the order and positioning of each piece afterwards. Currently, we have to perform this task manually. However, this is an easy task as it is just a matter of counting. The shapes are limited to sizes that are not large to facilitate this task. Our data is not affected as only local information in the data is extracted and used. We make several assumptions when extracting the data. We assume that the user adds each piece individually to the existing shape. This is typical for most of the pieces. If the user fabricates the overall shape by first building separate parts and then attaching them together, we do not consider those attachments (but we still extract data in the building of each separate part). We consider rules that are in the basic form. Each newly added piece can lead to multiple rules if it gets connected to multiple pieces. The data records the frequency of each rule, and no data is collected for the starting rules.

Additional Rewriting Rules. Our examples mainly demonstrate the use of the standard Lego bricks, as these are the most common pieces for human artists who manually build shapes from Lego. However, our framework is general and applies to other piece types. We demonstrate this with an example of another piece type that is often used: Lego slopes (examples in Figure 4). The rules in the basic form are simple but work well in practice. They are enough for our analysis as more complex rules can be reduced to the basic form. However, our formal framework is general, and we can define rules with multiple pieces in u and multiple newly added pieces in v (examples in Figure 4). The tradeoff for having these rules is that our analysis can be more efficient if the rules can be applied, but our analysis can be less efficient in situations where the rules are not likely to be used at all. We currently choose to not collect data for these rules because it is ambiguous to interpret the user's intention to allow for them.

4 Generation of Lego Pieces

Voxelization. Our algorithm first converts the input 3D model into a voxel representation. We can adjust the resolution of the voxel representation to change the size of the resulting physical shape. The volume occupied by each 1x1 block of a standard Lego brick is 8mm by 8mm by 9.6mm (width by length by height). The actual width and length are slightly smaller to facilitate the physical process of assembling the pieces, but these actual sizes do not affect our representation. Hence we voxelize the 3D model such that each 1 by

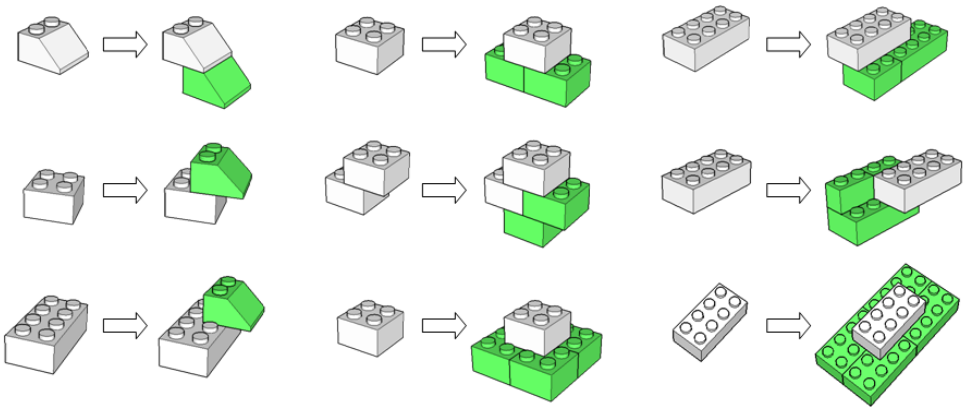

Figure 4: *Left column: Examples of rewriting rules with Lego slopes. Two right columns: Examples of more complex rules with multiple pieces in u and v. Each rule also specifies the order of the newly added pieces if there are two or more of them.*

1 by 1.2 unit is one voxel. Each rule can be rotated along the vertical axis to align with the horizontal axes to produce (up to three) more rules. In our implementation, these rotations are explicitly performed once (before structural analysis). For cases with Lego slopes, each 1 by 1 by 0.6 unit is one voxel. The standard bricks then have two layers of voxels, while the slopes have a slanting pattern of voxels.

Structural Analysis. This is the main step of our algorithm and it performs a backward tracing of the rewriting rules to give an ordered sequence of Lego pieces for assembling a connected shape. The backward tracing procedure builds a tree where each node has a grid of voxels. In our implementation, each node only stores the local changes of the voxels between that node and its parent node, in order to avoid using a large amount of storage. The tree's root node is initalized with the voxels from the input 3D model. We then take a depth-first approach to expand the tree's nodes. For each node, we iterate through the rewriting rules in random order and try to apply each in reverse. This backward tracing tries to find a subpattern in the current voxels that matches pattern v of a rewriting rule. If there is a match, we apply that rule in reverse by replacing the subpattern in the voxels with pattern u, and add a child node to the current node to store the new grid of voxels. The tracing procedure can backtrack through the tree if no rules can be applied at a node, and its child nodes have been expanded and no solution was found. If the grid of voxels becomes empty, we have reached a possible solution. If all nodes have been expanded and the grid of voxels was never non-empty, the input 3D model cannot be fabricated by our system (Σ, R). The solution is given by re-applying the rewriting rules in the forward direction starting from ε (Figure 5 shows a simple example).

There are some details needed in the above procedure that we describe here. First, before a rule is applied in reverse, we check that there is space below or above the new piece(s) and that the space is not completely enclosed such that the new piece(s) can physically be placed there. We do so by flood filling the empty space of the voxel grid with the padded piece (add half a voxel of space to each direction of the piece) until it reaches some boundary or it is completely enclosed. Second, each rule is labeled with an up or down direction to indicate from which direction the new piece is added. The step of finding a subpattern that matches v scans the voxel grid according to this direction, which makes the pattern matching more efficient and intuitive. Third, the pattern matching keeps track of the matched pieces so far, and a pattern match is only allowed (except for the first pattern match) if the newly added pieces can connect to the already matched pieces. This is particularly important for the efficiency of the algorithm.

Using Example Data in Rewriting Rules. For building shapes with a large number of pieces, we avoid the combinatorial explosion of states by using example data to reduce the branching factor of the tree. For each tree node, we do not try to expand all the rules as explained above. The frequency of rules in the collected data form a categorical distribution, and we select b rules by sampling from this distribution to test for expansion

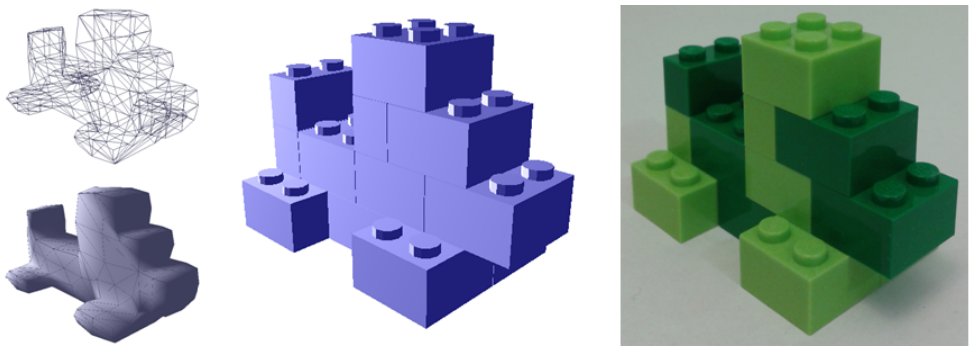

Figure 5: *Simple example generated with the rules in Figure 2. Two renderings of the input model are shown on the left. Applying the backward tracing procedure gives this backward order of rules: {10,5,6,5,8,9,3,2,2,2,5,5,0}. Applying these rules in the forward order generates the result in the middle. We built the physical shape with real Lego bricks (shown on the right).*

first. The branching factor b is adaptive such that it starts from a small number of rules and increases towards a larger number as the search gets closer to a solution (i.e. as the number of unmatched voxels gets closer to zero). The starting rules are always tested for expansion since they do not add any branch, as either no match exists or a solution is found. Since there is no data for the more complex rules, we pick one of them with equal probability to test for expansion once in a while (determined by parameter). Humans have a natural ability to consider the stability of the shape when connecting the pieces. We indirectly consider the shape's stability by using the example data. However, this is considered only locally as the rules themselves are local representations.

Hierarchical Approach. We use a hierarchical approach to reduce the depth factor of the tree. The original model is parsed into smaller shapes and structural analysis is applied to each shape. We use a simple parsing method, as there is already much work in segmentation and this is not our focus. We parse by cutting in horizontal planes after voxelization, where there are relatively big changes in the number of voxels between planes. The skyscraper example in Figure 6 demonstrates this. The advantage of this parsing is that the parts can easily be combined together as each part is connectable itself and there are typically many voxels (and at least one) connecting the parts. We choose this simple parsing method because segmenting before voxelization and/or cutting through horizontal layers can further complex the problem.

5 Results

Our largest alphabet Σ includes slopes and these types of bricks: 1x1, 1x2, 1x3, 1x4, 2x2, and 2x4. Our largest R has 223 rules (not including rules that are the same through rotations). Most of these are in the basic form and similar to the rules in Figure 2. In our collected data, the smallest user-built shape has 31 pieces and the largest shape has 116 pieces. The total frequency count is 2484, with the lowest being 1 and the highest being 139. Many of the rules are not common. The branching factor b typically starts at 10-30 and increases towards 100-200. As the search nears a solution, most of the rules tested for expansion do not match and the actual branching factor can still be small. For the complex rules, the parameter for picking one to test for expansion can be between once every fifth node to once every node.

Figure 6 shows input 3D models of a house, a skyscraper and a mug, and the corresponding results using our method. The runtime for the simple example in Figure 5 is within 0.1s, for the house and skyscraper are in the order of seconds, and for the space shuttle (Figure 1) and mug are in the order of minutes.

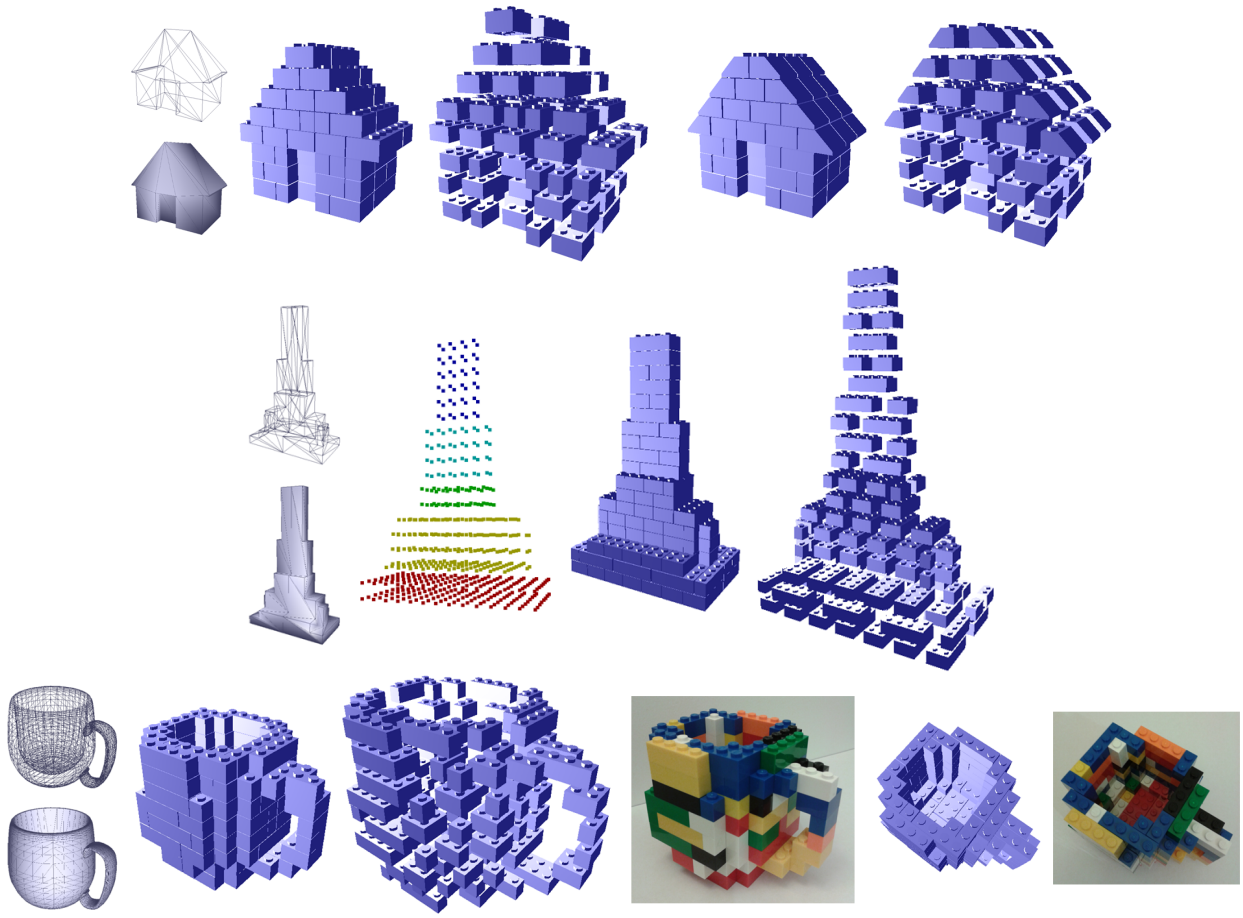

Figure 6: *Input 3D models (from Trimble 3D Warehouse and Princeton Shape Benchmark) and results generated by our algorithm. From top: house (61 bricks); house with slopes (same input with slope pieces allowed in (Σ, R), 43 bricks and 24 slopes); skyscraper (showing voxelization with five parsed parts through horizontal planes, 92 bricks); and mug (m504, with additional view from top of mug, 132 bricks). We use a simple method based on the pieces' positions to create the "exploded" view of the Lego pieces. The results give a sequence of pieces that is guaranteed to be buildable. Please see the video (https://tinyurl.com/47n7t4ub) for these sequences.*

6 Discussion

We have presented a formal grammar-based system and a search framework to take a 3D model as input, and generate as output an arrangement of Lego pieces that can be assembled together to form a physical object resembling the input model. We demonstrate our system with 3D models of varying complexity.

One limitation is that the grammar rules are manually created. In both human languages and programming languages, the grammar rules are often written by hand for the purposes of analyzing and understanding the languages. For future work, we may explore how to generate the grammar rules automatically. We may also explore more on how to get more complex rules and how to use them effectively. In addition, we can improve on how we automatically extract example data for the rules. We may use computer vision techniques and/or place sensors on the Lego pieces.

A limitation is that if a 3D model has very smooth and curved surfaces, or sharp non-axis-aligned edges, they cannot be represented well by the Lego pieces since their shapes are usually planar. But this is expected

of Lego, which is considered to be a fun representation of 3D shapes.

We currently do not take the colors of the shapes into account, and we focus on the geometry of the shapes. We could use bricks with different colors at the desired locations if colors are needed.

References

[1] M. Agrawala, D. Phan, J. Heiser, J. Haymaker, J. Klingner, P. Hanrahan, and B. Tversky. "Designing effective step-by-step assembly instructions." *ACM Transactions on Graphics*, vol. 22, no. 3, 2003, pp. 828–837.

[2] M. Bächer, B. Bickel, D. L. James, and H. Pfister. "Fabricating articulated characters from skinned meshes." *ACM Transactions on Graphics*, vol. 31, no. 4, 2012, pp. 47:1–47:9.

[3] A. Bedford. *The Unofficial LEGO Builder's Guide*. No Starch Press, 2012.

[4] M. Bokeloh, M. Wand, and H.-P. Seidel. "A connection between partial symmetry and inverse procedural modeling." *ACM Transactions on Graphics*, vol. 29, no. 4, 2010, p. 104.

[5] J. Calì, D. A. Calian, C. Amati, R. Kleinberger, A. Steed, J. Kautz, and T. Weyrich. "3D-printing of non-assembly, articulated models." *ACM Transactions on Graphics*, vol. 31, no. 6, 2012, pp. 130:1–130:8.

[6] X. Chen, A. Golovinskiy, and T. Funkhouser. "A Benchmark for 3D Mesh Segmentation." *ACM Transactions on Graphics*, vol. 28, no. 3, 2009, p. 73.

[7] X. Chen, H. Li, C.-W. Fu, H. Zhang, D. Cohen-Or, and B. Chen. "3D fabrication with universal building blocks and pyramidal shells." *ACM Transactions on Graphics*, vol. 37, no. 6, 2018, pp. 189:1–189:15.

[8] M. Davis, R. Sigal, and E. Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, 2nd ed. Academic Press, 1994.

[9] F. A. Fanni, A. Dal Bello, S. Sbardellini, and A. Giachetti. "Approximating Shapes with Standard and Custom 3D Printed LEGO Bricks." *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, 2021.

[10] R. A. H. Gower, A. E. Heydtmann, and H. G. Petersen. "LEGO: Automated Model Construction." *European Study Group with Industry, ESGI 32*, 1998.

[11] K. Hildebrand, B. Bickel, and M. Alexa. "crdbrd: Shape Fabrication by Sliding Planar Slices." *Computer Graphics Forum*, vol. 31, no. 2pt3, 2012, pp. 583–592.

[12] J.-Y. Hong, D.-L. Way, Z.-C. Shih, W.-K. Tai, and C.-C. Chang. "Inner engraving for the creation of a balanced LEGO sculpture." *The Visual Computer: International Journal of Computer Graphics*, vol. 32, no. 5, 2016, pp. 569–578.

[13] J. W. Kim, K. K. Kang, and J. H. Lee. "Survey on Automated LEGO Assembly Construction." *International Conference in Central Europeon Computer Graphics, Visualization and Computer Vision*, 2014, pp. 89–96.

[14] M. Lau, A. Ohgawara, J. Mitani, and T. Igarashi. "Converting 3D furniture models to fabricatable parts and connectors." *ACM Transactions on Graphics*, vol. 30, no. 4, 2011, pp. 85:1–85:6.

[15] S. Lee, J. Kim, J. W. Kim, and B.-R. Moon. "Finding an Optimal LEGO Brick Layout of Voxelized 3D Object Using a Genetic Algorithm." *Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 1215–1222.

[16] W. Li, M. Agrawala, B. Curless, and D. Salesin. "Automated generation of interactive 3D exploded view diagrams." *ACM Transactions on Graphics*, vol. 27, no. 3, 2008, p. 101.

[17] L. Luo, I. Baran, S. Rusinkiewicz, and W. Matusik. "Chopper: partitioning models into 3D-printable parts." *ACM Transactions on Graphics*, vol. 31, no. 6, 2012, pp. 129:1–129:9.

[18] S.-J. Luo, Y. Yue, C.-K. Huang, Y.-H. Chung, S. Imai, T. Nishita, and B.-Y. Chen. "Legolization: optimizing LEGO designs." *ACM Transactions on Graphics*, vol. 34, no. 6, 2015, pp. 222:1–222:12.

[19] D. Mathias, C. Snider, B. Hicks, and C. Ranscombe. "Accelerating product prototyping through hybrid methods: Coupling 3D printing and LEGO." *Design Studies*, vol. 62, 2019, pp. 68–99.

[20] J. McCrae, K. Singh, and N. J. Mitra. "Slices: a shape-proxy based on planar sections." *ACM Transactions on Graphics*, vol. 30, no. 6, 2011, pp. 168:1–168:12.

[21] P. Merrell and D. Manocha. "Continuous model synthesis." *ACM Transactions on Graphics*, vol. 27, no. 5, 2008, p. 158.

[22] J. Mitani and H. Suzuki. "Making papercraft toys from meshes using strip-based approximate unfolding." *ACM Transactions on Graphics*, vol. 23, no. 3, 2004, pp. 259–263.

[23] Y. Mori and T. Igarashi. "Plushie: an interactive design system for plush toys." *ACM Transactions on Graphics*, vol. 26, no. 3, 2007, p. 45.

[24] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. "Procedural modeling of buildings." *ACM Transactions on Graphics*, vol. 25, no. 3, 2006, pp. 614–623.

[25] S. Ono, A. Andre, Y. Chang, and M. Nakajima. "LEGO Builder: Automatic Generation of LEGO Assembly Manual from 3D Polygon Model." *ITE Transactions on Media Technology and Applications*, vol. 1, no. 4, 2013, pp. 354–360.

[26] Y. I. H. Parish and P. Müller. "Procedural modeling of cities." *ACM SIGGRAPH*, 2001, pp. 301–308.

[27] P. Petrovic. "Solving LEGO brick layout problem using Evolutionary Algorithms." *Norwegian Conference on Computer Science*, 2001.

[28] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, 1991.

[29] A. Rivers, A. Adams, and F. Durand. "Sculpting by numbers." *ACM Transactions on Graphics*, vol. 31, no. 6, 2012, pp. 157:1–157:7.

[30] G. Saul, M. Lau, J. Mitani, and T. Igarashi. "SketchChair: An All-in-one Chair Design System for End-users." *International Conference on Tangible, Embedded and Embodied Interaction (TEI)*, 2011, pp. 73–80.

[31] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. "The Princeton Shape Benchmark." *Shape Modeling International*, 2004, pp. 167–178.

[32] P. Song, C.-W. Fu, and D. Cohen-Or. "Recursive interlocking puzzles." *ACM Transactions on Graphics*, vol. 31, no. 6, 2012, pp. 128:1–128:10.

[33] B. Stephenson. "A Multi-Phase Search Approach to the LEGO Construction Problem." *International Symposium on Combinatorial Search*, vol. 7, no. 1, 2016, pp. 89–97.

[34] C. A. Vanegas, T. Kelly, B. Weber, J. Halatsch, D. G. Aliaga, and P. Müller. "Procedural Generation of Parcels in Urban Modeling." *Computer Graphics Forum*, vol. 31, no. 2pt3, 2012, pp. 681–690.

[35] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. "Instant architecture." *ACM Transactions on Graphics*, vol. 22, no. 3, 2003, pp. 669–677.

[36] H. Xu, K.-H. Hui, C.-W. Fu, and H. Zhang. "Computational LEGO technic design." *ACM Transactions on Graphics*, vol. 38, no. 6, 2021, pp. 196:1–196:14.

[37] G. Yun, C. Park, H. Yang, and K. Min. "Legorization with multi-height bricks from silhouette-fitted voxelization." *Computer Graphics International Conference*, 2017, pp. 40:1–40:6.

[38] J. Zhou, X. Chen, and Y. Xu. "Automatic Generation of Vivid LEGO Architectural Sculptures." *Computer Graphics Forum*, vol. 38, no. 6, 2019, pp. 31–42.