EUROGRAPHICS 2016 / J. Jorge and M. Lin
(Guest Editors)

Volume 35 (2016), Number 2

Building Construction Sets by Tiling Grammar Simplification

Javor Kalojanov[1,3] Michael Wand[2,3] Philipp Slusallek[1,3]

[1]Saarland University, [2]Mainz University,
[3]Intel VCI Saarbrücken

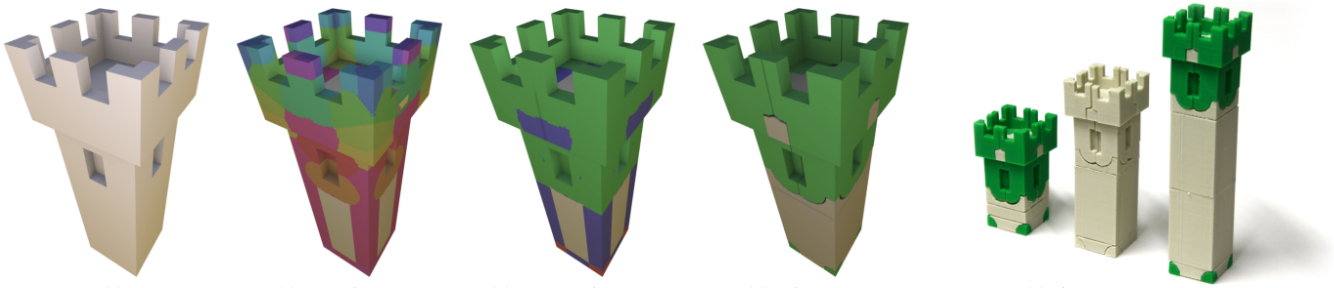

Figure 1: *We present a system for the fabrication of construction sets from example geometry. Pipeline overview: (1) input geometry (2) symmetry-based tiling grammar (3) optimized grammar (4) pieces that can be manufactured and assembled to (5) produce shape variations.*

Abstract

This paper poses the problem of fabricating physical construction sets from example geometry: A construction set provides a small number of different types of building blocks from which the example model as well as many similar variants can be reassembled. This process is formalized by tiling grammars. Our core contribution is an approach for simplifying tiling grammars such that we obtain physically manufacturable building blocks of controllable granularity while retaining variability, i.e., the ability to construct many different, related shapes. Simplification is performed by sequences of two types of elementary operations: non-local joint edge collapses in the tile graphs reduce the granularity of the decomposition and approximate replacement operations reduce redundancy. We evaluate our method on abstract graph grammars in addition to computing several physical construction sets, which are manufactured using a commodity 3D printer.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

The motivating problem for our paper is is to create physical construction set games from virtual example geometry: Given a suitable 3D model, we want to fully automatically fabricate sets of building blocks (for example, with a 3D printer) of a small number of different types that can be assembled to recreate not only the example model but also many shape variations. Informally, one could characterize this problem as "inverse LEGO": computing suitable bricks from example models.

What are the necessary and desirable properties of a 3D constructor set? Obviously, the pieces have to be manufacturable, and must assemble into the example shape(s). The second requirement has two aspects. First, it should be possible to segment the input model into the constructor pieces without creating overlaps. Second, a disassembly sequence of rigid motions must exist, allowing to construct the resulting shapes from the manufactured pieces. Good building blocks should be generic: We expect to obtain only a very small number of different *types of building blocks*, that already construct the geometry considered. From the perspective of the human user, this facilitates understanding of how to utilize the pieces. In this work, we will obtain these restricting requirements by trading-off another desirable quality – the expressiveness of the construction set, i.e., the amount of variations that can be created using the pieces.

Of course, making a good "construction set game" has many

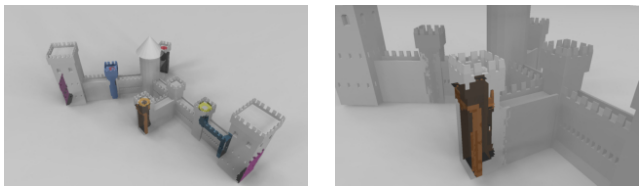

Figure 2: *Tiles from Bokeloh et al. 2010 [BWS10]. Some individual parts are highlighted, showing a redundant segmentation (e.g.: concentric circular pieces on the tower tops in orange / red), and irregular shape of the pieces.*

more facets, such as further constraints on building block geometry (size variability, aspect ratios, minimum size of connecting surfaces, unique connector geometry to aid assembly, etc.) or psychological aspects such as the relation between "fun to play", or "challenge" and geometric properties. However, our paper focuses only on the abstract problem of obtaining compact sets of physically realizable building blocks, which we perceive to be the core obstacle at this point.

The construction set construction problem is tightly related to recent techniques for inverse procedural modeling [ŠtávaBM*10, BWS10, KBW*12, JTRS12, ZCOM13, MWZ*13]. These methods decompose 3D objects into building blocks and derive rules for their assembly. New shape variations are created by assembling these pieces along matching boundaries. While the concrete definition of matching and ability to connect differs, the process can be understood as building a graph with building blocks as nodes that connect along edges according to fixed rules. Such systems for assembling graphs of pieces are denoted as *tiling grammars* [Ber66, KBW*12, LVW*15], and our construction sets are a special instance of these.

Inverse modeling for *physical fabrication* brings a novel problem to our attention that has not been considered in virtual scenarios: the *quality of the decomposition* in terms of geometry and complexity. Existing approaches often yield oddly shaped building blocks with complex assembly rules, which makes fabrication and manual assembly very hard. Figure 2 shows an example composition of a castle model from Bokeloh et al. [BWS10], consisting of oddly shaped tiles and redundant rules (such as the need for the yellow and brown adaptor pieces in Figure 2). A more general approach is discussed by Kalojanov et al. in follow up work [KBW*12]: It introduces "r-microtiles" as building blocks that construct the whole family of locally similar shapes. In this case, the construction even yields infinitesimal pieces (points and curve segments), in addition to complexly shaped finite parts, rendering physical manufacturing fully impossible (see Figure 3).

To address these problems in a principled way, our paper introduces the concept of *simplifying tiling grammars*. We start with a overly complex tiling grammar and determine an improved set of only a few, easy to assemble building blocks by following a variational approach: We minimize an objective function that trades-off the complexity of the grammar (shape, size, and ease of assembly, as captured by the number of tiles and rules) with the amount of shape variations it still represents. Building block assemblies are represented as a graph and iteratively transformed graph by collapsing related subsets of its edges, modifying the assembly rules accordingly. Further, pieces of related geometry but different type according to the grammar rules can be replaced by a single piece type using an approximate shape matching approach. We employ a Monte Carlo search technique to sample the space of possible decompositions and select the best one according to our cost function. Finally, we generate final 3D-manufacturable building blocks using solid modeling operations.

For our motivating application, the fabrication of construction games, we use Kalojanov et al.'s "r-microtile" as input to the simplification pipeline. Both the locality and the restriction to rigid transformations of their model capture the essential goals of finding locally cohesive pieces that can be manually assembled in a construction game. In addition, we also conduct a short experiment with generic (non-rigid) discrete graph grammars from manual labeling [LVW*15] as input, demonstrating that also more generic, abstract tiling grammars can be handled, too.

We apply our method to a number of polygonal input scene and print tiles using a commodity 3D printer. In our experiments, we could successfully create construction set games for a range of example models; the main restriction here is that the input models themselves need to be well-defined solids, compatible with available constructive solid modeling tools. We should also stress that our paper focuses on the problem of obtaining a small number of easy-to-assemble building blocks; we only consider the minimal set of mechanical constraints necessary to generate a set of 3D realizable pieces with collision-free assembly.

In summary, our paper makes two main contributions: First, it already presents a simple, fully-automatic system for creating a sets of physically manufacturable, and easy-to-assemble building blocks from suitable example 3D shapes. The complexity of the construction sets is traded-off against attainable shape variability and can be flexibly controlled. Second, on the technical side, we introduce the notion of simplifying tiling grammars (in particular, this entails simplifying the building blocks of a shape, i.e., an encoding of its partial symmetry structure) and present a first practical algorithm for this task. Being based on a generic variational formulation, we believe that the method could be easily adapted for related tasks and could have further applications in inverse-procedural modeling and symmetry-based shape processing.

2. Related Work

One aspect of our work is the computation and analysis of a set of pieces that assemble a 3D shape. Mitra et al. [MYY*10] analyze the functional properties of man-made assemblies. Guo et al. [GYL*13] introduce a framework that can automatically compute and visualize the assembly and disassembly of man-made objects. Their work together with a wide range of research on disassembly analysis and planning [DA03] address the problem of finding optimal assembly/disassembly sequences. In this paper we try to automatically generate an assembly, for which at least one assembly sequence exists. This is a complicated problem if general

types of interlocking parts and assembly motions are considered. Some existing approaches like [SFCO12, SG99] restrict the types of interlocking components and only allow assembly via translational motions. In this paper, we instead use a soft optimization constraint to conservatively minimize the amount of interlocking components, which can prevent assembly.

Xin et al. [XLF*11] and Song et al. [SFCO12] address the problem of decomposing a 3D shape into a set of interlocking puzzle pieces. The results of our method can be used to compute 3D puzzles, however we do not enforce a decomposition into interlocking constructor pieces, and instead of restricting the assembly actions to translations, we allow for arbitrary rigid motions. We also base our work on a symmetry analysis method, which allows to compute building blocks that assemble variations of the input shape. The works by Fu et al., Singh et al., and Eigensatz et al. [FLHCO10, SS10, EKS*10] consider the problem of decomposing surfaces into sets of equivalent patches. The authors optimize the shape of a polygonal mesh in order to minimize the types of polygons necessary to represent the model. In this paper we compute triangulation invariant building blocks of an arbitrary three-dimensional shape. Similarly to the previous related works, the authors do not consider construction of a collection of related shapes in addition to the original.

While also not the main focus of this work, symmetry detection plays an important role in the preprocessing stage. See e.g. the STAR report by Mitra et al. [MPWC13] for a comprehensive overview. We detail the technical aspects of our symmetry detection method in a recent technical report [Kal15]. The symmetry analysis method by Wang et al. [WXL*11] considers a second order structure in addition to the pieces of a symmetry-induced shape decomposition (also see [MWZ*13]) focus on structure- preserving editing of the input. In contrast, we are interested in modifying the computed structure and optimizing it for a given application.

The construction of shape grammars from partial symmetry has been introduced by Bokeloh et al. [BWS10]. The authors find tiles by considering pairs of non-intersecting symmetric cuts and obtain a context-free grammar by rejecting overlapping cuts. The representation obtained is not manufacturable and often overly complex and oddly shaped (Figure 2). Kalojanov et al. [KBW*12] generalize the concept by "microtiling" the surface through the set of all possible symmetric curve pairs, which yields a prohibitively complicated and non-physical grammar (including infinitesimally small, "continuous" pieces).

Liu et al. [LVW*15] extend the notion of rigid tilings to generic graphs of partially matching, deformable tiles, where only the topology of the assembly is constrained and free-form deformation is used for assembly. We use this generalized model in an experiment to demonstrate simplification of tiling grammars independent of rigid symmetry. The lack of rigidity restricts this setting to virtual modeling (fabrication is not possible). Further, Liu et al. note that even simple rigid tiling grammars are Turing-capable, thereby making it impossible to fully assess shape variability (even assembling finite tile sets is NP-hard). We therefore rely on counting symmetric cut-pairs [BWS10] to lower-bound the variability of the generic

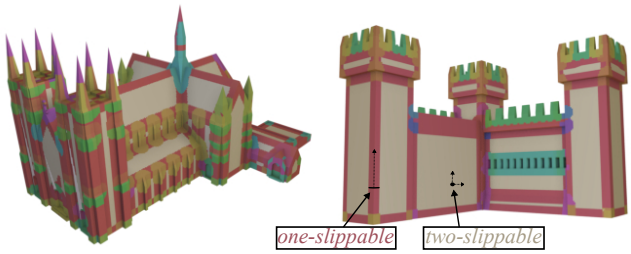

Figure 3: *Two models are decomposed into their microtiles w.r.t. r-symmetry. Tiles of the same shape have the same color. Gray area indicates infinite sets of 2-slippable microtiles (each of which is a single point) that form planar segments. The red pieces along the edges of the model visualize 1-slippable geometry (the theoretical tiling consists of the set of all orthogonal cross-sections). Slippable tiles arise from continuous (partial) symmetry.*

tiling grammars. In our system, the hard problem of assembly is left to the human user (this is the objective of the game), but the negative complexity results again clearly show that we must proactively bound the complexity of the decomposition. Finally, the negative complexity results also justifies our stochastic search strategy; any exact solution would be infeasible.

Construction of optimal shape grammars has previously been explored by Wu et al. [WYD*14] and Talton et al. [TYK*12], also trading-off compactness (a coding length model) against expressiveness. The main difference to our work is that only context-free shape grammars are considered, i.e., all assembly rules must be hierarchically structured. In the context of our construction set application, this restriction is unnatural and makes automatic grammar generation difficult. Further, there is a close connection between tiling grammars and partial symmetry: Their building blocks directly provide a low-level encoding of redundancy in shapes. We believe that further processing and simplifying this type of information is a valuable tool of its own, given the importance of partial symmetry in modern geometry processing algorithms.

3. Structure from Partial Symmetries

We now discuss our processing pipeline. The first step is to create a building block decomposition. As previously discussed, we employ the method of Kalojanov et al. [KBW*12, Kal15], using the code of the latter reference. For completeness, we briefly review how this approach generates tiling grammars from partial symmetry:

The main idea is to find partial symmetries in shapes, and cut shapes such that symmetric geometry is identified as the same type of building blocks.

***r*-symmetry:** Given a 3D triangle mesh, two points are considered *r-symmetric* if their respective spherical neighborhoods of radius r are identical up a rigid mapping (that matches the two points). r-symmetry forms an equivalence relation among surface points.

***r*-similarity:** A shape S_1 is r-similar to S_2 if every point from S_1 is r-symmetric to a point on S_2. This means, S_1 is made of the same local features as S_2. Our objective is to build tiles out of which we can assemble many shapes, all of which are r-similar to a specific piece of reference geometry, which exemplifies the style of the shape family.

Building blocks: Conceptually, we obtain r-microtiles by computing all r-symmetries between all pairs of points. The points that are always mapped together (i.e., which are associated with the same set of symmetry transformations) belong to the same tile. Hence, the tiling yields a segmentation of the input geometry into a disjoint set of regions that are connected to neighboring tiles through their boundaries (Figure 3).

Types of tiles: We will observe that many tiles have the exact same geometric shape. These tiles are said to have the same *type*, i.e., the type denotes the shape of the tiles up to a rigid motion.

Slippable tiles: Because of continuous (partial) symmetry (slippable surfaces [GG04]), we can obtain infinitesimally small tiles: In triangle meshes, any 1-slippable tile is a curve segment extruded along a straight linear edge. The curve is orthogonal to its extrusion direction. 2-slippable tiles are just points within planar regions (see Figure 4).

Initial segmentation: We obtain the initial segmentation as a r-microtile decomposition (typically $r = 1$-3% of the scene diameter). Regions of continuous symmetry are not decomposed into infinitesimal tiles but kept as whole pieces, just tagged as having either one or two degrees of freedom. They will be further decomposed later in the process.

Finally, r-microtiles can build all shapes $r+\varepsilon$-similar to the input for $\varepsilon > 0$ [KBW*12]. This flexibility usually causes them to be very detailed and unsuitable for fabrication. For example, the cathedral model in Figure 3 is decomposed into 931 microtiles of 527 different types (excluding slippabale geometry, which is just shown as in uniform gray and red, respectively).

3.1. Tiling Graphs from r-Microtiles

We obtain a graph representation of the microtile decomposition by simply recording neighborhood relations in the previously computed decomposition (Figure 4). Every output region is represented by a node in the graph, and two nodes are connected if and only if they have an overlapping spherical r-neighborhood, still treating slippable areas as single tiles. We also record the type of each (non-slippable) tile.

Implicit grammar encoding: Importantly, the example graph itself implicitly encodes the tiling grammar that describes a whole family of shape. Formally, we consider any object a valid shape variant that (i) consists only of rigid copies of tiles (nodes) of the example graph, and (ii) connections to neighboring tiles are only permitted if the original graph shows at least one example of the two tiles connecting with the same relative transformation across the same pair of boundary curves.

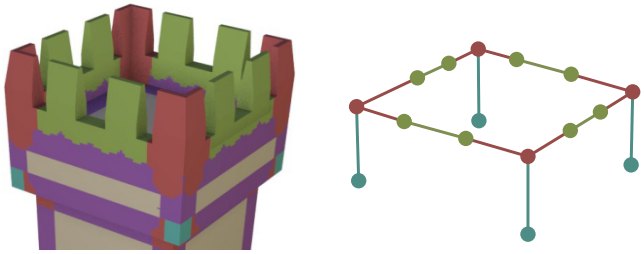

Figure 4: *Mircotile decomposition (left) and the resulting graph (right). Equivalent building blocks, graph nodes, and graph edges are colored identically. One- and two-slippable areas (purple and gray) are omitted from the graph.*

3.2. Generic Discrete Tiling Grammars

Simplifying rigid r-microtile grammars for fabrication is the main focus of our paper. Nonetheless, the framework of simplifying tiling grammars is more generally applicable. In order to demonstrate this, we also consider the model of Liu et al. [LVW*15] as potential input: In this work, the authors use a painting interface to annotate tile types on surfaces, which do not need to be rigidly symmetric. From these annotations, we again obtain graphs of nodes representing different types of shapes. Also in this case, the shape grammar is encoded implicitly by an example graph, where each node carries a type annotation and consistently interfaces with its neighborhood through boundaries. From the point of view of graph topology, there is no difference to the rigid case (it is even easier as the special case of slippable tiles is absent). To conduct experiments with this model in our paper, we simply perform manual labeling of a microtile in a non-rigid way (ensuring fixed boundary rules for each type during setup); see Figure 12.

4. Graph-based Grammar Simplification

We now address our main objective of simplifying tiling grammars. We assume that we are given an (abstract) graph of tiles and define a cost function, which aims to quantify how well the underlying decomposition is suited for 3D manufacturing. We later optimize the segmentation according to the cost model by modifying the graph.

4.1. A Cost Model for 3D Manufacturing

Our goal is to minimize the complexity of the tiling grammar by reducing the number of different pieces and make them easier to manufacture and assemble. To this end, we define a number of objective functions that characterize this complexity and perform a stochastic minimization. Importantly, as the graphs encode a shape grammar, we will now perform simplification operations simply on the example graph, which implicitly changes the grammar.

Redundancy: Type annotations in the graph capture redundancy: by recognizing geometry of the same type, storing the same geometry repeatedly is avoided. Specifically, the initial r-microtile

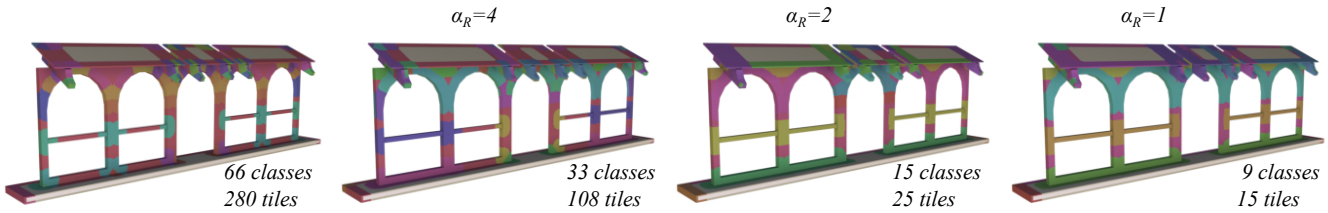

Figure 5: *A microtile decomposition of a fence model (left) and optimization examples using a basic cost funtion $Cost_1 := \alpha_R C_R + C_C$ to control the amount of simplification. From left to right: C_R rises from 0.1 to 0.4; C_C drops from 1 to 0.14.*

decomposition is optimal in the following sense: Each different r-neighborhood is provably contained in exactly one microtile type, hence the surface area of all r-symmetric points is never represented redundantly. We compute the surface area n_i covered by one instance per tile class i (technically, we approximate area by counting non-empty voxels produced by the decomposition algorithm), as well as the total area in the scene N_V. Our first objective is thus to avoid reintroducing redundancy during the simplification process. This is expressed as:

$$C_R := \max\left\{\frac{\sum_i n_i}{N_V}, \alpha_{SA}\right\}, C_R \in [0,1] \tag{1}$$

which approximates the fraction of non-redundant surface area in the representation. We cut out the cost below the constant α_{SA} to relax the penalty and allow introducing a small amount of redundancy. We set this parameter to $\frac{1}{20}$ throughout our experiments.

Simplicity: It is well understood that one of the most important guidelines for (manual) design for manufacturability and assembly is to aim at reducing the number of types of pieces necessary to construct a given object [Boo96]. This translates to minimizing the number of different tile types. When used with alternative means of fabrication (such as injection molding instead of 3D printing), this can even reduce production costs significantly. We use the initial microtile decomposition as a benchmark for the shape complexity and express the cost for the number of microtile classes as a fraction of this initial amount (Figure 5). Let $TC_{current}$ and $TC_{initial}$ be the set of microtile classes for the optimized and the initial decomposition. We define:

$$C_C := \frac{|TC_{current}|}{|TC_{initial}|}, C_C \in (0,1]. \tag{2}$$

Assemblability: Being able to ensure that once physical pieces are produced it will be possible to put them together is a complicated problem – the blocks might have a complex form obstructing insertion of neighboring pieces. To our knowledge, there is no related work in the field of geometric modeling dealing with this problem for pieces that can have arbitrary shape. Instead, related works (e.g. [XLF*11, SFCO12]) restrict the allowed shape of their puzzle pieces such that assemblability can be ensured a priori. We therefore use insights from the field of mechanics and design for manufacturing to reduce the likelihood of assembly problems during optimization.

In their work on mechanical disassembly analysis, Shyamsundar et al. [SG99] identify that invalid assemblies are the result of *interlocking sub-assemblies* with at least 2 parts. The latter are characterized by the *interfering volumes* of the parts participating in the assembly. An interfering volume is defined by the intersection of a piece with the convex hull of a neighboring piece. Motivated by this, we penalize the overlap between bounding boxes of neighboring microtiles. This reduces the amount and size of the interfering volumes of neighboring pieces and hence minimizes the chance for interlocking sub-assemblies (see Figure 6). We use the term:

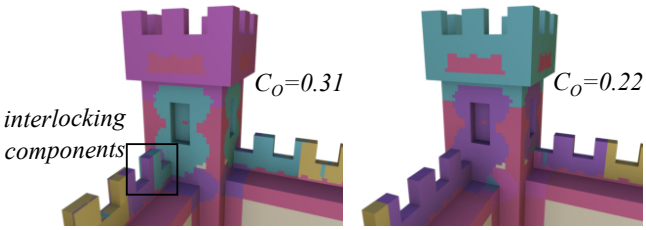

Figure 6: *The higher the overlap cost (C_O), the greater is the chance of producing interlocking components that can cause problems during assembly.*

$$C_O = \frac{1}{N_T}\sum_{i=0}^{N_T}\frac{\sum_{j=0}^{N_T}\frac{V_{ij}}{V_i}\delta_{ij}}{\sum_{j=0}^{N_T}\delta_{ij}}, C_O \in [0,1], \tag{3}$$

where V_{ij} is the volume of the intersection of the bounding boxes of tile i and tile j, V_i the volume of the bounding box of tile i, N_T is the total number of tiles and

$$\delta_{ij} := \begin{cases} 1 & \text{if tiles } i \text{ and } j \text{ are adjacent} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The denominator $\sum_{j=0}^{N_T}\delta_{ij}$ is the number of neighbors of tile i and allows to normalize the maximum value of the cost to 1. Note that both C_O and δ_{ij} are defined w.r.t. the decomposition of the shape into microtiles instances as opposed to C_R and C_C, which are based on the number of types (classes) of pieces.

Global Cuts: The previous two objectives encourage simplicity; a naïve optimization would yield the whole input model as single building block. We therefore trade this off against shape variability. As detailed later, in Section 4.5, shape variability is lower bounded by the number of matching pairs of r-symmetric cuts. Cutting the shape along those symmetry boundaries allows to re-join the result-

ing pieces in more than one way, thereby creating shape variations. We capture this by the cost term:

$$C_{GC} := |n_{target} - n_{cuts}|, C_{GC} \in [0, \infty], \quad (5)$$

quantifying the difference between a (user defined) desired (n_{target}) and the actual number (n_{cuts}) of symmetric global cuts. Global cuts guarantee means of shrinking and expanding the model, i.e., changing the number of tiles; additional variability is due to exchanging of tiles of different type but matching boundary.

Finally, in our tests we used the sum of the above cost functions to measure the quality of the microtile graphs we computed

$$Cost := \alpha_R C_R + C_C + C_O + C_{GC} \quad (6)$$

The weight α_R and the target number of global cuts can be used to control the amount of simplification during optimization. Figure 5 illustrates how α_R influences the trade off between the two main terms C_R and C_C. Scaling the term C_O in order to prevent complicated assembly is also possible, but was not necessary for our test models. If not said otherwise, we set $\alpha_R = 1$ vary only the number of global cuts in the remaining experiments.

Please note that our framework in principle is also fairly general; the variational formulation makes it easy to define, in future work, alternative objective functions for other applications.

4.2. Simplification Operations

We now consider operations that incrementally reduce the complexity of the tiling grammar. Specifically, we consider two types of moves: A non-local joint edge collapse that combines tiles of different types, and a replacement operation that expresses tiles of slightly different geometry by same type (i.e., by fixed geometry).

4.2.1. Edge Collapses

We first consider merging of tiles through an non-local joint edge-collapse in the graph: A single edge collapse will merge two instances of building blocks with each other. In order to preserve the symmetry structure in the new representation, we only collapse whole equivalence classes. That is, if we collapse an edge between a tile x of type X, and tile y of type Y we collapse all edges connecting pairs of tiles of type X and Y (connected through the specific matching edges) in the graph. This is important because it prevents reintroducing redundancies through the operation.

Note that collapsing a set of graph edges does not change the input geometry; it only coarsens its segmentation into microtiles. With regards to the geometry of the input object, this is a lossless operation. Also note that although we try to improve the actual building blocks, we are able to efficiently perform transformations directly on the microtile graph.

4.2.2. Tile Replacement

The ability of edge collapses to preserve the original shape and its symmetry information makes the algorithm sensitive to inaccuracies in the initial decomposition. The latter can be caused due to numerical errors, noise or other types of deformations in the input geometry.

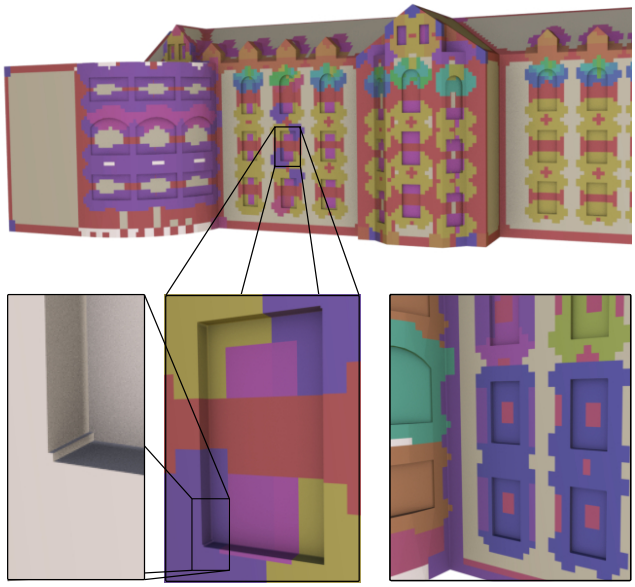

Figure 7: *The corners of several of the windows of this building differ from the rest, apparently due to a modeling inaccuracy. Lower right: merging the microtiles into larger building blocks allows to approximately match these regions despite the discrepancies.*

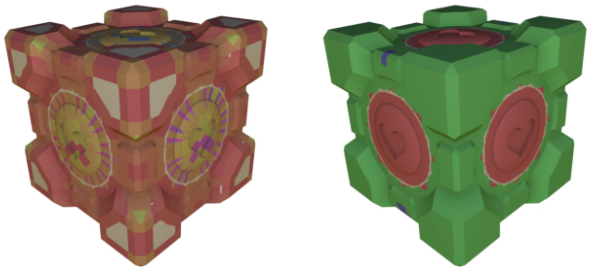

Figure 8: *Replacing tile classes during simplification allows to fix errors in the initial decomposition. In the right example the parts of the model with complex curvature are identified as similar, even though some of their r-neighborhoods do not match precisely (left).*

On the other hand, our graph simplification approach allows the discovery of redundant regions in a hierarchical fashion. We can take advantage of this property by matching pairs of tile classes after merging them. We perform approximate matching by dividing each shape into voxel cells [Kal15], and computing a set of a plane equations (fragments) per cell. Planes are then matched by comparing normals and offsets, tolerating deviations in normal directions up to 10° and offsets up to typically 1/512 of the model diameter. Increasing the size of the segments reduces the importance of local errors or deformations. Although not necessary in general, our similarity metric scales with the size of the regions being matched. This allows to identify geometric regions of arbitrary size that are

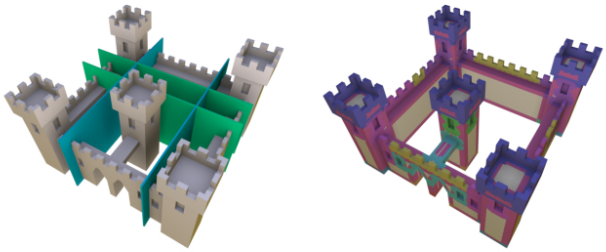

Figure 9: *An example model and two pairs of symmetric planar cuts (left) and an optimized microtile decomposition that preserves these cuts (right). We do not merge microtiles across cuts of this type to increase the number of shape variants we can construct.*

identical up to a user-defined area (we used 10% in all tests). When computing the initial decomposition the size of each region is fixed – a single r-neighborhood. Hence, matching merged sets of building blocks allows to distribute the local errors over larger surface area and discover additional redundancies in the simplified decomposition.

Including tile replacement operations during the optimization made our algorithm robust to local deformations caused by artist errors (Figure 7) and discretization artifacts in curved shapes in some models (see Figure 8).

4.3. Optimization Approach

How do we schedule the different operations for finding an optimal tiling grammar? Let *the edge cost* be defined as the difference in the graph cost after collapsing this edge (and all edges equivalent to it). Unfortunately, the cost of a given edge is not invariant under graph transformations. This is the case because edge collapses change global properties of graph such as the number of tile classes (C_C) or introduces changes in the node connectivity (δ_{ij}). This makes the behavior of the cost function for the graph edges difficult to predict and does not allow to use an efficient optimization strategy.

On the other hand, for a fixed set of edges to be collapsed, the resulting graph is the same regardless of the order in which the operations are performed. This reduces the search space: Instead of looking for an optimal sequence of transformations, one only needs to determine an optimal subset of the graph edges to collapse.

The individual graph operations do not have very high computational costs. This facilitates performing a rather exhaustive, randomized search in reasonable time. We perform a series of random graph transformations – edge collapses and tile replacement attempts. Each series is terminated randomly via Russian roulette with probability $\frac{1}{N_C}$ (where N_C is the number of different node types). For each model, we perform about N_C searches and keep the graph with the best cost. Although not offering strong optimality guarantees, this approach already achieves significant improvements with a moderate sample size. A side note: For applications like design for manufacturing, results will likely be judged subjectively. Here, being able to provide multiple good solutions via sampling constitutes a certain advantage.

Designing our system, we have also experimented with several greedy, deterministic strategies. For example, we collapsed the largest set of equivalent edges. A single step of this algorithm introduces the smallest decrease in redundancy and keeps C_R (see Equation 1) smallest while at the same time reduces the number of building blocks and potentially the classes of microtiles (C_C, Equation 2). This method was faster compared to the random search, which delivered slightly superior results. The best cost for the model in Figure 9 we obtained with the greedy search was 0.8, compared to 0.68 from random sampling.

4.4. Shape Variations

We consider the manufacturing related properties of the building blocks such as complexity and ability to assemble to be more important in our test scenario, as long as some of the initial variability is preserved. Therefore, we opted to trade-off some of the expressive power of the building blocks to further improve the other properties expressed by the cost function (see Equation 6). To achieve this we allow three operations that can potentially reduce the number of shape variations:

- We merge neighboring microtile pairs from the same class, which can affect the size variations of grid-like repetitive patterns.
- We collapse cycles of equivalent edges, if any of the edge in the cycle has to be collapsed. This can reduce variations in the size of the possible microtile cycles.
- We construct and collapse edges through one-slippable regions of the input shape. This can restrict production of r-similar shapes through stretching or shrinking the corresponding edges.

If the above operations are disallowed and tile replacement is not performed during simplification, the expressiveness of the tiling grammar will not be reduced. In our evaluation, we also encountered cases, where the initial set of shape variations was preserved even after performing some of the "unsafe" simplification steps.

4.5. Global Cuts

How can we quantify variability? Unfortunately, tiling grammars are known to be able to encode the computation of general Turing machines [Ber66, LVW*15], rendering it impossible to understand in general any of their properties algorithmically, including their expressiveness. For this reason, we resort to a lower bound for quantifying their potential for shape variations: We recall the observation of Bokeloh et al. [BWS10] that a sufficient (but not always necessary) condition for shape variations is the presence of pairs of global cuts that match each other (r-similarly) under a fixed rigid transformation.

Before the graph optimization step, we set a desired number of such global cuts that the final graph should contain (we used 2 or

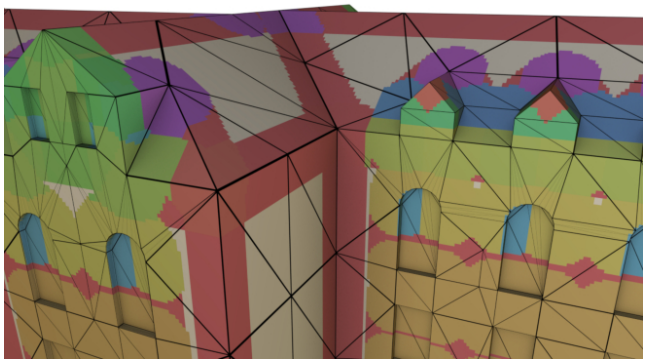

Figure 10: *A triangle mesh is colored with the detected partial symmetries using a voxel grid. Neither of the two discretizations provides building blocks invariant under rigid transformations.*

3 in our experiments). During graph transformations we keep track of how many pairs of cuts remain valid after the operation is performed and penalize deviation of the target number in the cost function with the term C_{CG} (see Equation 5). If the number of remaining global pairs of cuts becomes less or equal to the target quantity, we stop performing graph transformations that eliminate the existing cuts. In other words we set a hard constraint on the minimal number of global symmetric cuts.

The cuts computed by Bokeloh et al. [BWS10] are computed in a voxelized representation (see Figure 2), permitting complex curves. For 3D printing, this representation is not suitable. While it would be possible to represent these curves analytically, this leads to significant challenges for the later solid modeling that is required for 3D printing. For this reason, we opt for a simple but robust solution and restrict the additional global cuts to *planar* cuts only (see Figure 9).

The global planar cuts are also used to presegment the slippable geometry that we so far have considered as single coherent pieces: Each global plane is constructed to separate pairs of rigid tiles either indirectly, through a connecting slippable edge, or directly, through the middle of a non-slippable connecting edge. This simplifies the detection of the global cuts: we need to check once for each non-slippable graph edge and sample along the remaining edges to exhaust all possible cut locations. Choosing to split rigid edges in a ratio 1:1 also simplifies the later extraction of the non-slippable building blocks via CSG operations. In the further process, this also splits intersecting planar (2-slippable) nodes. In the supplemental document to this paper, we provide more technical details on how global cuts affect the final, volumetric building blocks.

5. Volumetric Building Blocks

Our analysis so far is performed on a triangle mesh representing an input surface. In this section, we describe another contribution of this paper – our method for converting the optimized surface decomposition into manufacturable building blocks. The main challenge we need to address, is ensuring that the resulting pieces are invariant under rigid transformations. As shown in Figure 10, the correspondences based on point-wise r-symmetry yield triangulation independent decomposition, and a straight-forward voxelization in rectilinear voxels does not deliver interchangeable rigid pieces. This prevents the mass-production of the individual constructor pieces. In a scenario, where 3D-printing is used to manufacture the pieces, the problem presents itself during assembly, where only a correct variant of a building block will fit given configuration of adjacent pieces.

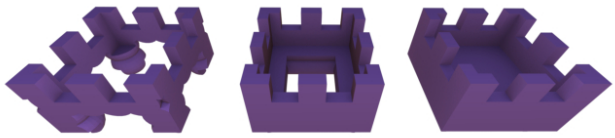

Figure 11: *The spherical r-neighborhoods of corners (left) are merged with one-slippable edges (middle) and two-slippable planar areas (right) of the model and then carved out of the input model using CSG operations.*

The naïve solution to only identify pieces as similar if they are triangulated in the same way will obviously discard most of the initial symmetries and significantly reduce the possible shape variations. To address the problem of computing exactly matching rigid building blocks we derive a discretization from the definition of r-symmetry. Even though the initial correspondences are defined per point on the input surface, we can prove that it suffices to consider a finite amount of surface elements in order to obtain a complete decomposition of corresponding points.

We consider three types of discrete elements: geometric corners, geometric edges and planar regions. In other words, we separate the points on the input surface based on their r-slippability. Each point on the triangle mesh will either be two-slippable, or inside the r-neighborhood of a corner or edge, which shows that the input surface is covered by r-neighborhoods of finitely many elements apart from the two-slippable regions, which are easy to extract.

Using the above considerations, we can enclose all important neighborhoods by a set of spheres around corner features and cylinders along the line features. This allows to extract a 3D manufacturable representation of our r-symmetric building blocks via CSG operations with simple geometric primitives. In our implementation, we compute a CSG tree that encloses the r-neighborhood of each building block. Intersecting the tree with the input model delivers 3D manufacturable building blocks. We verify our extraction method, using a Fused Decomposition Modeling (FDM) 3D-printer to produce a set of constructor pieces for several test models.

We also implemented several strategies to resolve intersections of r-neighborhoods on tile boundaries. Although not essential for the evaluation of our method, this ensures a canonical representation of the final pieces and allows for simplified extraction. This

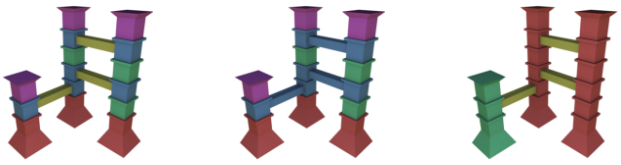

Figure 12: *Left: A model is decomposed manually in 18 pieces of 5 types. Middle: The optimized version consists of 12 pieces of 4 types. Right: Allowing replacement of approximately matching tiles further reduces the complexity to 6 pieces of 3 types.*

	# rigid pieces			# types of rigid pieces			cost		
	init.	opt.	red.	init.	opt.	red.	init.	opt.	red.
tower	104	8	13×	15	2	7×	1.21	0.28	4×
bridge	232	8	29×	27	3	9×	1.39	0.26	5×
house	246	27	9×	36	11	3×	2.19	0.42	5×
castle	693	149	4×	53	12	4×	1.31	0.65	2×
church	2042	8	255×	309	4	77×	1.32	0.5	2×
asylum	1130	81	14×	277	17	16×	3.32	0.26	12×

Table 1: *The improvement in the rigid building blocks and the overall cost of the decomposition we achieved after optimization.*

is the case, because that shape of the extracted building blocks remains the same, regardless of the order in which the pieces are "cut out" of the input model. We discuss the technical details of this method in a supplementary document provided along with this paper.

6. Evaluation

We demonstrate that our approach is practical by implementing all stages of the pipeline depicted in Figure 1. The cutting of exact manufacturable building blocks from the original shape is done by generating a script and executing it in OpenSCAD [Ope] - a tool for programmatic CAD modeling.

Decomposition Quality: We tested our method on several meshes of various complexity displayed in Figure 13, 14, 15 and 16. Note that we do not refer to the triangle count, but the number of partial self-symmetries, when discussing the complexity of the input model (our method is fully agnostic to the triangulation). Our test models are subject to self-symmetries generated by a large number (in the order of $10^4 - 10^5$) of transformations. Using graph transformations, we could improve both the cost for the decomposition by more than an order of magnitude and the number of building blocks by more than two orders of magnitude (see Table 1). The resulting building blocks are still able to reproduce the major part of the meaningful r-similar variations of the input object (see Figures 13, 14, 15, 16, and 18).

We also tested our grammar simplification method on a manually decomposed shape, similar to test models used by [LVW*15]. We could improve the complexity of the decomposition even for the already simple example in Figure 12. We show the optimal results without and with tile replacement operations (Section 4.2.2), because the latter allow to identify pieces marked by the user as different. Note that this test did not use the cost terms C_O and C_{GC}; the corresponding properties (assembly and variability) were provided by the user.

Global cuts: Instead of computing the complete shape decomposition (Figures 13, 14, and 15), the last step of the pipeline can be modified to generate constructor pieces by splitting the shape along its global cuts (Figures 16 and 18). In general, the global cuts provide only a subset of the shape variations. Individual rigid pieces can also be swapped, e.g. the tower tops in Figure 15(4). Finally, the initial expressiveness of the shape grammar can be preserved by restricting the graph simplification operations, which coincidentally is the case for the final decomposition of the tower and bridge models in Figures 1 and 13.

3D Manufacturing: In order to demonstrate that the decomposition we compute is manufacturable, we used a Fused Decomposition Modeling 3D printer (MakerBot Replicator 2 [Mak]) to produce the resulting pieces. Although we use a soft constraint to ensure the assemblability of the building blocks, our method works well in practice: penalizing the overlap of the bounding volumes of neighboring pieces was sufficient and we did not encounter problems when assembling our test models. It was also possible to assemble our test models even if the one and two-slippable pieces were printed together (see Figure 17 and the supplemented video).

Complexity parameters: We also explore how we can adapt the granularity of our construction sets by varying parameters (Figures 5,19, and the supplemental document): Increasing the weight α_R for the redundancy term C_R in the cost formulation limits the amount of simplification and allows to preserve more of the initially discovered partial symmetries. This increases the amount of shape variations. In Figure 19, the use of larger α_R enables swapping the types of castle towers, modifying the length of the castle walls, as well as more variations of the facade of the asylum model. The difference between α_R and the target number of cuts n_{target} w.r.t. creating shape variations is that not all related shapes preserved by α_R can be constructed using manufactured, rigid building blocks. Variations that involve slippable geometry are problematic. For example, swapping the types of windows in a facade after the building blocks are manufactured is only possible if both pieces have the same silhouette. This makes the use of α_R better suited for modeling virtual shapes.

Performance: All computations were performed on a PC with an Intel Core i7-3770K CPU, 16 GB of RAM, and an NVidia GeForce 570 GTX graphics card. The initial symmetry detection took about 1 minute for the models in Figures 14 and 15. The random search of a tile graph with optimal cost took up to 20 minutes, most of which was spent on matching tile replacement candidates. This part of the implementation can be made interactive, by providing feedback in sub-second times after a small number of graph transformations and allowing the user to accept or discard the modifications. Using OpenSCAD to perform the CSG operations and compute the triangle meshes for the manufacturable building blocks was the slowest part of the process. We do not consider this to be an important performance bottleneck, since each

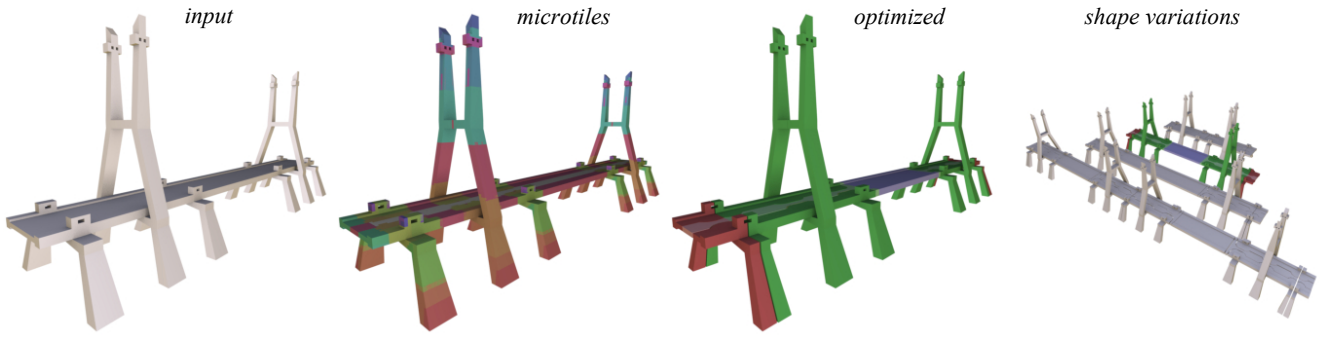

Figure 13: *A bridge is initially decomposed into 232 microtiles of 27 types, excluding slippable elements. The optimized decomposition with two global cuts consists of 13 pieces of 5 types including 5 slippable elements of 3 types. The initial cost of 1.39 was reduced to 0.26. We created some of the possible shape variations with the simplified building blocks.*

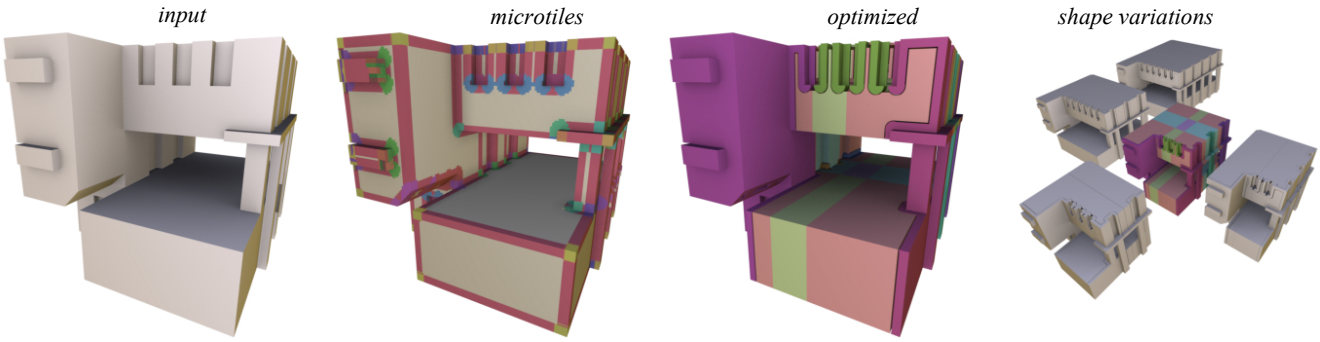

Figure 14: *A model of a modern building is initially decomposed into 246 microtiles of 36 types, excluding slippable elements. The optimized decomposition consists of 47 pieces of 28 types including 20 slippable elements of 17 types. The initial cost of 2.19 was reduced to 0.42. We preserved two global cuts and used them to construct some shape variations.*

operation has to be performed as a post-process and only once for each type of rigid building blocks and additional 2^n times for the slippable pieces, where n is the number of pairs of symmetric cuts. Therefore, we opted for highly tessellated meshes for the building blocks, resulting in run-times between several minutes and 8 hours for the most complex set of slippable pieces of the castle model (Figure 15).

7. Limitations and Future Work

The shape analysis approach presented in this chapter has several limitations that provide interesting avenues for future work.

Our system for building construction sets still remains rather basic. We only consider the minimal amount of mechanical properties necessary to create the final constructor pieces: we use a soft constraint to ensure assemblability and neglect the structural strength of the pieces apart from parameterizing the thickness of the two-slippable elements in the CSG scripts that generate them. Higher-level aspects, such as assembly aids or the human reception of construction games are not considered yet at all.

Further, we also still limit the shape of the global symmetric cuts used to generate shape variation in the slippable building blocks to infinite planes. Splitting along cuts of an arbitrary shape is not easy to perform with the elementary CSG primitives and operations we used in our tests. In general, we found in particular the robustness of constructive solid geometry operations to be a key limitation in practice. In addition of problems with handling complexly structured Boolean operations, the inability of the CSG library to process some 3D models was the prime reason for not being able to handle input meshes within our system. There has been work on robust and efficient CSG algorithms (see e.g. [BEH*05]) using techniques such as adaptive interval arithmetics, the incorporation of which would be an interesting topic of future work.

Because the method we describe here is in essence a simplification algorithm, the amount of shape variations and the possible improvement in the decomposition depend on the model and its initial segmentation. Even though approximately matching simplified building blocks extends the space of possible solutions, which made the system more robust to inaccurately modeled objects, not every shape presents a symmetry structure that can be simplified into an interesting set of building blocks.

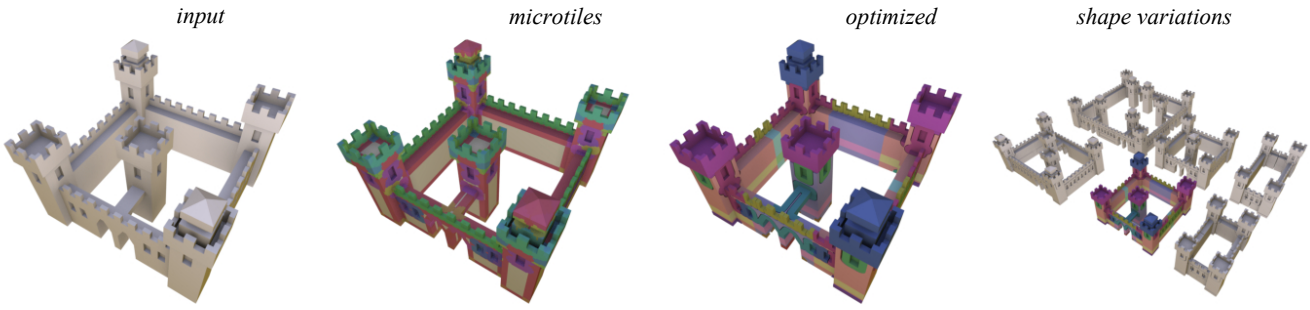

Figure 15: *A castle is initially decomposed into 693 microtiles of 53 types, excluding slippable elements. The optimized decomposition consists of 191 pieces of 39 types including 42 slippable elements of 27 types. The initial cost of 1.31 was reduced to 0.65. In addition to the three global cuts, we created shape variations by exchanging rigid pieces.*

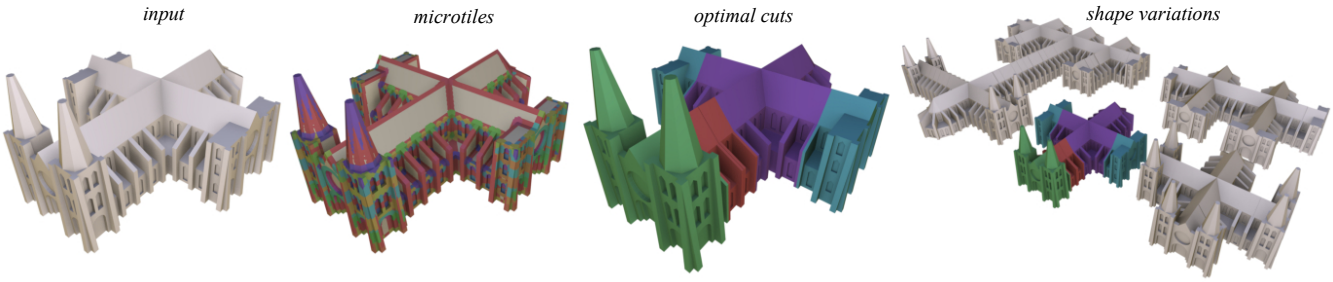

Figure 16: *A church is initially decomposed into 2042 microtiles of 209 types, excluding slippable elements. The optimized decomposition consists of 8 pieces of 4 types excluding slippable elements. The initial cost of 1.32 was reduced to 0.5. We only used the discovered global cuts to decompose the model and did not separate rigid and slippable microtiles.*

8. Conclusion

In this paper, we have introduced the idea of simplifying tiling grammars. A tiling grammar encodes shape families as an assembly of pieces by docking their boundaries according to prescribed rules. Our algorithm takes an overly detailed description, as typically arising from fully automatic inverse procedural modeling, and coarsens the model to a user specified level while maintaining the variability of the encoded shape family. The core idea is to optimize the representation by trying many combinations of merging and replacement operations to minimize an objective function that trades-off complexity and a lower bound on retained variability.

We employ this new algorithm to construct a system for fabricating 3D construction set games fully automatically from a 3D model with controllable complexity. The larger system is obtained by applying the simplification algorithm to building blocks obtained from rigid microtiling, which itself yields flexible but overly complicated tiling grammars. The simplified decomposition is converted through a CSG construction into printable and easy to assemble pieces that are fabricated on a 3D printer.

We believe that the idea of simplifying tiling grammars could become a useful tool for broader applications, in particular in the context of inverse procedural modeling (where model quality is difficult to control). Our framework is easy to extend in terms of both adding further simplification operations as well as formulating various objective functions. This paper already shows that manufacturing constraints (overlap penalties) can be easily integrated, and exploring further applications would be a very interesting opportunity of future work.

9. Acknowledgments

The cathedral, church, and asylum models were downloaded from Thingiverse [Thi]. We would like to thank their creators for sharing their work. We thank Silke Jansen and Han Liu for discussion. This work has been supported by the Intel Visual Computing Institute (IVCI) at Saarland University.

References

[BEH*05] BERBERICH E., EIGENWILLIG A., HEMMER M., HERT S., KETTNER L., MEHLHORN K., REICHEL J., SCHMITT S., SCHÃŰMER E., WOLPERT N.: Exacus: Efficient and exact algorithms for curves and surfaces. In *Proc. of the 13th Annu. European Sympos. Algorithms (ESA'05)* (2005). 10

[Ber66] BERGER R.: The undecidability of the domino problem. *Memoirs of the American Mathematical Society 66* (1966). 2, 7

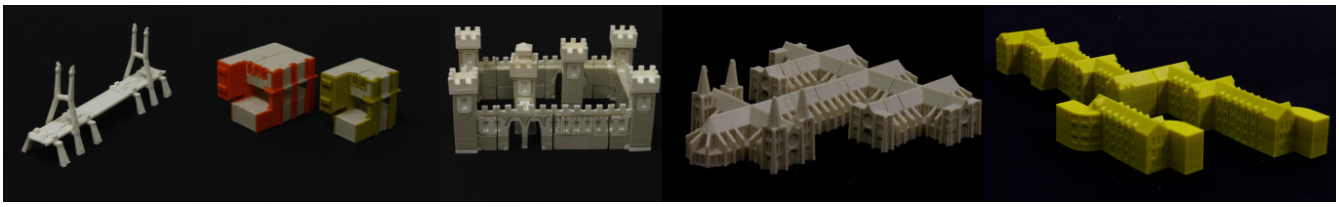

Figure 17: *Our test models assembled from 3D printed building blocks. We used a Makerbot Replicator 2 to print all models. We manually removed the pair of horizontal global cuts when manufacturing the castle model (third example) to simplify the assembly and only used global cuts to split the church and asylum models (the two rightmost examples).*

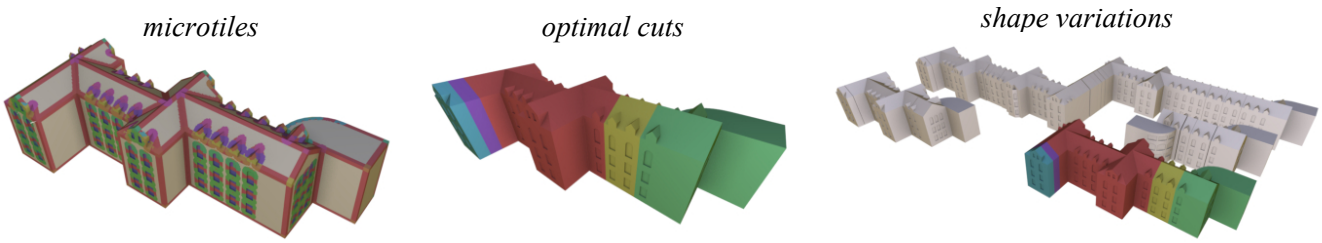

Figure 18: *An asylum building is initially decomposed into 1130 microtiles of 277 types, excluding slippable elements. The optimized decomposition consists of 81 pieces of 17 types excluding slippable elements. The initial cost of 3.32 was reduced to 0.26. We only used the discovered global cuts to decompose the model and did not separate rigid and slippable microtiles.*

[Boo96] BOOTHROYD G.: Design for manufacture and assembly: The boothroyd-dewhurst experience. In *Design for X*, Huang G., (Ed.). Springer Netherlands, 1996, pp. 19–40. 5

[BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph. 29* (July 2010), 104:1–104:10. 2, 3, 7, 8

[DA03] DONG J., ARNDT G.: A review of current research on disassembly sequence generation and computer aided design for disassembly. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 217*, 3 (2003), 299–312. 2

[EKS*10] EIGENSATZ M., KILIAN M., SCHIFTNER A., MITRA N. J., POTTMANN H., PAULY M.: Paneling architectural freeform surfaces. *ACM Transactions on Graphics 29*, 4 (2010), 45:1–45:10. 3

[FLHCO10] FU C.-W., LAI C.-F., HE Y., COHEN-OR D.: K-set tilable surfaces. *ACM Transactions on Graphics 29*, 4 (July 2010), 44:1–44:6. 3

[GG04] GELFAND N., GUIBAS L.: Shape segmentation using local slippage analysis. In *Proc. Symp. Geom. Processing* (2004). 4

[GYL*13] GUO J., YAN D.-M., LI E., DONG W., WONKA P., ZHANG X.: Illustrating the disassembly of 3d models. *Computers and Graphics 37*, 6 (2013), 574 – 581. Shape Modeling International (SMI) Conference 2013. 2

[JTRS12] JAIN A., THORMAHLEN T., RITSCHEL T., SEIDEL H.-P.: Exploring shape variations by 3d-model decomposition and part-based recombination. *Computer Graphics Forum 31*, 2pt3 (2012), 631–640. 2

[Kal15] KALOJANOV J.: *Efficient r-Symmetry Detection for Triangle Meshes*. Tech. rep., Saarland University, 2015. 3, 6

[KBW*12] KALOJANOV J., BOKELOH M., WAND M., GUIBAS L., SEIDEL H.-P., SLUSALLEK P.: Microtiles: Extracting building blocks from correspondences. *Computer Graphics Forum 31*, 5 (2012), 1597–1606. 2, 3, 4

[LVW*15] LIU H., VIMONT U., WAND M., CANI M.-P., HAHMANN S., ROHMER D., MITRA N. J.: Replaceable substructures for efficient part-based modeling. *Computer Graphics Forum* (2015). 2, 3, 4, 7, 9

[Mak] MAKERBOT:. http://www.makerbot.com. 9

[MPWC13] MITRA N. J., PAULY M., WAND M., CEYLAN D.: Symmetry in 3d geometry: Extraction and applications. *Computer Graphics Forum 32*, 6 (2013), 1–23. 3

[MWZ*13] MITRA N. J., WAND M., ZHANG H., COHEN-OR D., BOKELOH M.: Structure-aware shape processing. In *EUROGRAPHICS State-of-the-art Report* (2013). 2, 3

[MYY*10] MITRA N. J., YANG Y.-L., YAN D.-M., LI W., AGRAWALA M.: Illustrating how mechanical assemblies work. *ACM Trans. Graph. 29*, 4 (July 2010), 58:1–58:12. 2

[Ope] OPENSCAD:. http://www.openscad.org. 9

[SFCO12] SONG P., FU C.-W., COHEN-OR D.: Recursive interlocking puzzles. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 128:1–128:10. 3, 5

[SG99] SHYAMSUNDAR N., GADH R.: Geometric abstractions to support disassembly analysis. *IIE Transactions 31*, 10 (1999), 935–946. 3, 5

[SS10] SINGH M., SCHAEFER S.: Triangle surfaces with discrete equivalence classes. *ACM Trans. Graph. 29*, 4 (July 2010), 46:1–46:7. 3

[ŠtávaBM*10] ŠTÁVA O., BENEŠ B., MĚCH R., ALIAGA D. G., KRIŠTOF P.: Inverse procedural modeling by automatic generation of L-systems. *Computer Graphics Forum 29*, 2 (2010). 2

[Thi] THINGIVERSE:. http://www.thingiverse.com. 11

[TYK*12] TALTON J., YANG L., KUMAR R., LIM M., GOODMAN N., MĚCH R.: Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (2012), UIST '12, ACM, pp. 63–74. 3

[WXL*11] WANG Y., XU K., LI J., ZHANG H., SHAMIR A., LIU L., CHENG Z., XIONG Y.: Symmetry hierarchy of man-made objects. In *Proc. Eurographics* (2011). 3

[WYD*14] WU F., YAN D.-M., DONG W., ZHANG X., WONKA P.: Inverse procedural modeling of facade layouts. *ACM Trans. Graph. 33*, 4 (July 2014), 121:1–121:10. 3

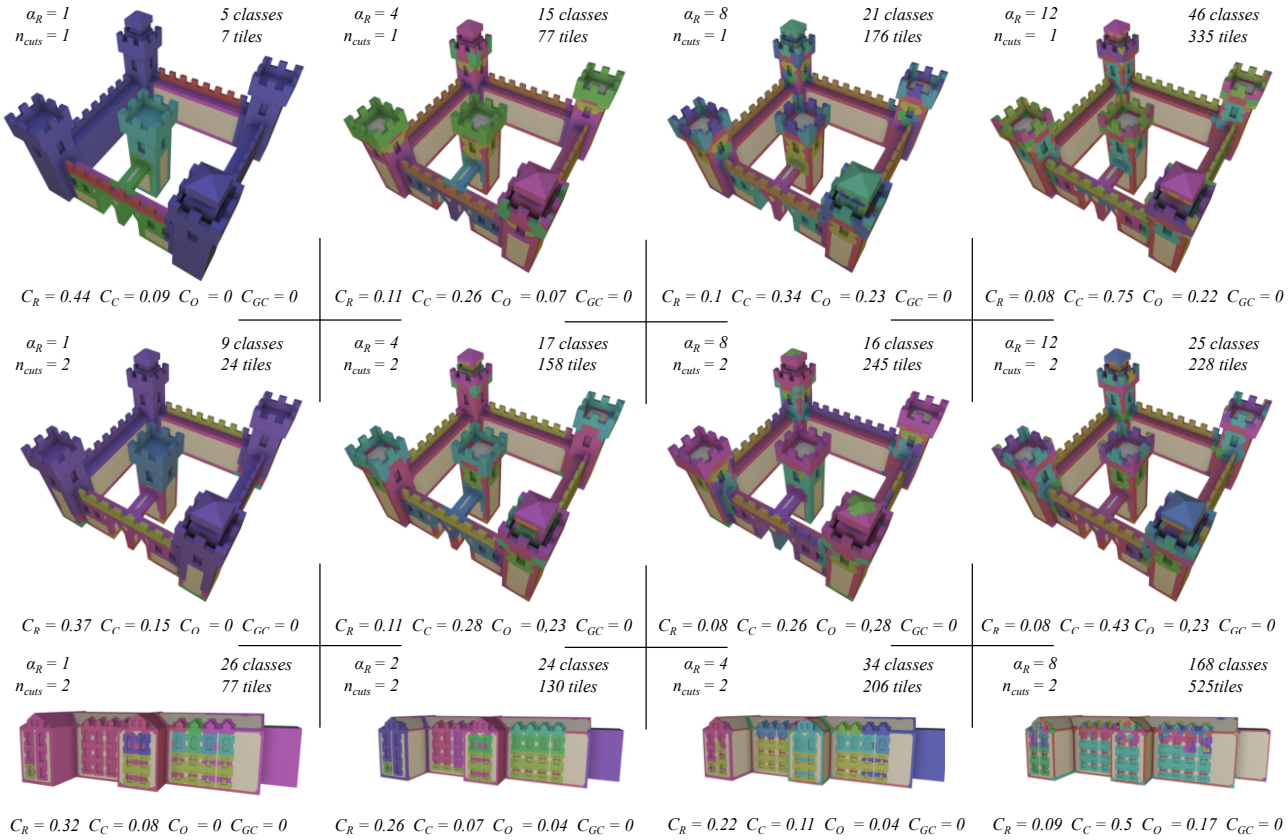

Figure 19: *The granularity of the building blocks can be controlled by weighting the redundancy term: C_R with α_R. Preserving more of the initially discovered symmetries can increase the amount of shape variations like the length of the castle walls or the possible facades of the asylum building. On the other hand, the smaller α_R the better the overall quality of the decomposition for manufacturing.*

[XLF*11] XIN S., LAI C.-F., FU C.-W., WONG T.-T., HE Y., COHEN-OR D.: Making burr puzzles from 3d models. *ACM Trans. Graph. 30*, 4 (July 2011), 97:1–97:8. 3, 5

[ZCOM13] ZHENG Y., COHEN-OR D., MITRA N. J.: Smart variations: Functional substructures for part compatibility. *Computer Graphics Forum (Eurographics) 32*, 2pt2 (2013), 195–204. 2