r-Symmetry for Triangle Meshes: Detection and Applications

Javor Kalojanov

Saarbrücken, August 2016

A dissertation submitted towards the degree
Doctor of Engineering Science (Dr.-Ing.)
of the Faculty of Mathematics and Computer Science
of Saarland University

UNIVERSITÄT
DES
SAARLANDES

Dean

Prof. Dr. Frank-Olaf Schreyer

Date of Colloquium

January 31, 2017

Examination Board:

Chairman

Prof. Dr. Joachim Weickert

Reviewers

Prof. Dr.-Ing. Philipp Slusallek

Prof. Dr. Michael Wand

Prof. Dr. Niloy J. Mitra

Scientific Assistant

Dr. Richard Membarth

Abstract

In this thesis, we investigate a certain type of local similarities between geometric shapes. We analyze the surface of a shape and find all points that are contained inside identical, spherical neighborhoods of a radius r. This allows us to decompose surfaces into canonical sets of building blocks, which we call **microtiles**. We show that the microtiles of a given object can be used to describe a complete family of related shapes. Each of these shapes is locally similar to the original, meaning that it contains identical r-neighborhoods, but can have completely different global structure. This allows for using r-microtiling for inverse modeling of shape variations and we develop a method for shape decomposition into rigid, 3D manufacturable building blocks that can be used to physically assemble shape collections. We obtain a small set of constructor pieces that are well suited for manufacturing and assembly by a novel method for tiling grammar simplification: We consider the connection between microtiles and non-context-free tiling grammars and optimize a graph-based representation, finding a good balance between expressiveness, simplicity and ease of assembly. By changing the objective function, we can re-purpose the grammar simplification method for mesh compression. The microtiles of a model encode its geometrically redundant parts, which can be used for creating shape representations with minimal memory footprints. Altogether, with this work we attempt to give insights into how rigid partial symmetries can be efficiently computed and used in the context of inverse modeling of shape families, shape understanding, and compression.

Kurzzusammenfassung

Diese Dissertation beschäftigt sich mit den Eigenschaften von Symmetrieabbildungen von geometrischen Flächen. Wir betrachten r-symmetrische Punkte, die mittels euklidische Transformationen aufeinander abgebildet werden können, sodass alle Nachbarpunkte in einer sphärischen Umgebung mit Radius r identisch bleiben. Wir zerlegen 3D-Modellen in Bausteine, die wir **Microtiles** nennen. Alle Punkte im selben Baustein haben die gleichen Symmetrietransformationen. Wir zeigen, dass sich aus den Microtiles eines 3D-Objektes einen wohl-definierten Raum ähnlicher Objekten konstruieren lässt. Jedes neues Objekt ist punkweise gleich mit dem ursprnglichen Modell, kann aber eine beliebige globale Struktur haben. Diese Eigenschaft ermöglicht das automatische Zerlegen von Geometrieobjekten in herstellbare Bausteine, die Objektvariationen bilden können. Wir stellen ein Optimierungsverfahren vor, wodurch wir verschiedene Eigenschaften von den Bausteinen beeinflussen können. Wir betrachten die Microtiles als eine formale Grammatik und transformieren einen darauf basierenden Graphen. Das ermöglicht uns anwendungspezifische Eigenschaften der Bausteine wie Variationsmenge, Einfachheit, Zusammensetzbarkeit zu verbessern. Dasselbe Optimierungsverfahren findet zusätzliche Anwendungen im Bereich der Datenkomprimierung. Da die Microtiles Redundanzen in der Geometrie darstellen, ist es möglich aus den Bausteinen eine Flächendarstellung zu berechnen, die möglichst wenig Speicherplatz benötigt. Im ganzen, gewinnen wir mit dieser Arbeit einen Einblick in die Zusammenhnge zwischen Symmetrien, der Analyse und Synthese von 3D-Objekten, und deren Komprimierung.

Acknowledgments

I would like to thank my supervisors Philipp Slusallek and Michael Wand. Philipp for being enthusiastic and supportive throughout my long stay in his department, and Michael for introducing me to Geometric Modeling. I am grateful to Niloy Mitra, who helped find a new research direction by inviting me to a summer school on 3D manufacturing and agreed to review the thesis on a short notice.

A nice side effect of the work on this thesis was the opportunity to meet and collaborate with wonderful people like Martin Bokeloh and Silke Jansen from Max-Planck Institute Saarbrücken, and my colleagues in the department of Computer Graphics at Saarland University: Stefan Popov, Iliyan Georgiev, Tomáš Davidovič, Lukáš Maršálek, Beata Turoňová, Piotr Danilewski, Felix Klein, Vincent Pegoraro, Arsène Pérard-Gayot, and Richard Membarth.

I am grateful for the limitless patience and support I receive from my family – my parents, my sister, my cousin, and Nadya. Together with several close friends, they made difficult times fly by.

Finally, I would like to dedicate this thesis to my grandparents, whom I robbed of the last opportunities to be together in order to work on this project.

Contents

1 Introduction

The similarities in geometric shapes, as well as the ability to recognize them, play an important role in the way humans perceive and understand their surroundings. Symmetries and regularities help us understand, memorize, and analyze objects and processes. We often use visual cues such as the observed similarities in the appearance of objects, to transfer other properties like purpose, functionality, and even estimate potential risk associated with them. For example, L-shaped cylindrical levers are often used as door-handles (purpose) and are activated by rotating them (functionality), while many plants have spikes that can hurt us. We even consider symmetric biological forms like faces to be more appealing, and possibly being a characteristic trait for good genetic material.

Not surprisingly, the mathematical principles related to shape similarities, or symmetries, have received a lot of attention from scientists, while at the same time artists have been using them as means to influence the aesthetics of their creations.

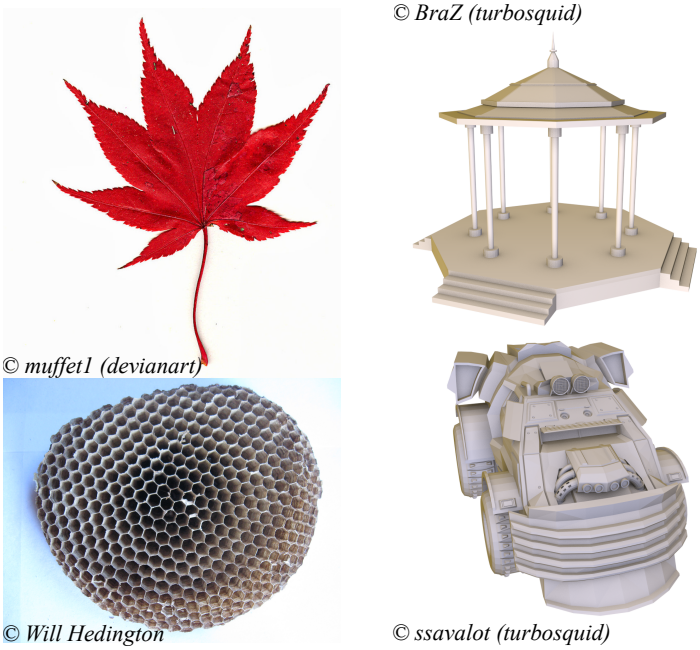


Figure 1.1: *Examples for symmetries occurring in nature (the Japanese maple leaf and the wasp nest) and in man made shapes (the gazebo and the concept truck).*

In computer graphics, symmetries can have multiple applications. The infor-

mation about geometric redundancy represented by the symmetric parts of an object can be used to simplify or compress it. The resulting representation can in turn be used for understanding certain properties related to the structure of the shape. We will demonstrate how such analysis allows for generating shape variations via inverse modeling. Other applications where symmetries play an important role include shape matching, shape segmentation, shape retrieval, and geometry completion.

In this thesis, we analyze a specific type of geometric self-correspondences, which we call r-symmetry. In Chapter 3, we first prove analytically that the surface of a geometric object can be decomposed into **microtiles** – a set of building blocks, out of which one can construct not only the original model, but a whole family of related (r-similar) shapes. Each of the new shapes is locally similar to the primary exemplar, but has a different global structure. These results are published in [KBW+12].

Based on this theoretical foundation, in Chapter 4 we derive algorithms for efficient and robust decomposition of triangle meshes into microtiles. The main challenge here is converting the initial point-wise correspondences into a finite set of building blocks with arbitrary shape and exact boundaries that are independent of the shape triangulation. The methods are discussed in [Kal15].

An additional contribution of this thesis is a method for tiling-grammar simplification, discussed in Chapter 5. Each similar shape constructed from microtiles is also a word from a language, generated by a context-sensitive grammar represented by the microtiles and a set of assembly rules given by their pairwise adjacency in the original surface. We analyze and simplify this grammar by computing an optimized decomposition according to a cost model. By using different objective functions we can optimize the initial shape decomposition for various applications.

We demonstrate the usefulness of the approach with two application scenarios. First, in Chapter 6, we use the geometric redundancies encoded by the building blocks for compressing triangle meshes. This is possible by representing the original mesh by a small set of pieces that are replicated in space. Second, in Chapter 7 we develop a method for automatic shape decomposition into 3D manufacturable constructor sets for modeling shape variations. The methods in Chapter 5 and Chapter 7 are discussed in [KWS16].

The author of this thesis has been the principle investigator for the research leading to the contributions listed above and detailed in Chapters 3, 4, 5, 6, and 7. The work has been conducted under the supervision and in collaboration with Prof. Michael Wand and Prof. Philipp Slusallek.

2 | Symmetry in 3D Geometry

In this chapter we introduce the mathematical notions upon which our work is founded. The readers with background in (partial) symmetry analysis in 3D can safely skip to Chapter 3.

2.1 Classical Theory

In general, the term **symmetry** is used to describe invariance under certain operations. For example, cars have mostly symmetric left and right exterior parts, meaning that mirroring the car along a vertical plane trough the middle will not change its appearance. In mathematics, this invariance of the entire shape for given transformations makes it possible to enclose the operations in algebraic **groups** [SS64].

Formally, we say that a geometric shape $S \subset \mathbb{R}^3$ is **symmetric** under a transformation T if S remains unchanged after applying T on it, i.e. $S = T(S)$, or $\forall p \in S, \exists q \in S : q = T(p)$. This implies that the set of transformations that preserve S forms a group: $(\{T|S = T(S)\}, \cdot)$ is a group. A set of objects, here transformations, and a composition operation $\cdot$ is a group if the following group axioms are satisfied:

- **Closure:** For every pair of elements T_1, T_2, $T_1 \cdot T_2$ is also inside the set. This holds in the case of symmetry transformations, since each of them leaves the defining shape S unchanged, hence $S = T_2(S) = T_2(T_1(S))$.
- **Identity:** There is an identity element I such that for any other element T in the set $I \cdot T = T = T \cdot I$. Trivially, the identity transformation (leaving S as is) exists for every object S.
- **Inverse Element:** For every transformation T in the group, there must exist an inverse T^{-1} such that $T \cdot T^{-1} = T^{-1} \cdot T = I$. For this to hold, for each of the considered operations transforming points in space, there should be another one (not necessarily different from the original), which returns each transformed point back to its original location.
- **Associativity:** Composition of more than two elements yield the same result regardless of priority, i.e. $(T_1 \cdot T_2) \cdot T_3 = T_1 \cdot (T_2 \cdot T_3), \forall T_1, T_2, T_3$. This follows analogous to the closure: The transformations leave S unchanged regardless of the priority of application.

In this thesis we will focus on certain kinds of geometric transformations such

as rotations, translations, mirroring, and combinations thereof. These particular types of transformations form the Euclidean group (or $E(3)$) and are referred to as **rigid** transformations or **isometries**. They have the property of preserving distances and angles, which is important for the analysis of typical man-made, non-deformable (rigid) objects.

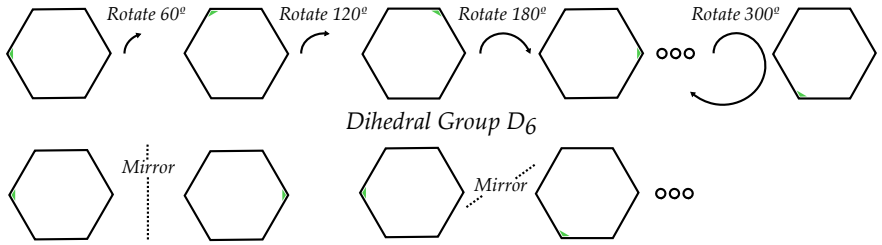


Figure 2.1: *A regular n-sided polygon has n rotational and n reflection symmetries, making for $2n$ elements in the dihedral group D_n.*

The classical theory for symmetries analyses the characteristics of the set of transformations mapping a geometric shape to itself exactly. Example shapes with such properties include regular polygons (see Figure 2.1). The most notable property of the generating transformations is that they form a closed algebraic group. Because the entire object remains unchanged after a transformation, we will call these types of symmetries **global**.

In our examples so far we have only considered transformation groups of finite elements - rotations and reflections. These operations preserve the center of the transformed object. This does not hold for translations. However, there are symmetry groups that do contain translations. Excerpts[1] of such groups are used to describe 1D, 2D repetitive patterns often observed in architectural ornaments.

Global symmetries are important for 3D graphics, and are by now a well understood topic thanks to their connection to mathematical groups. However, humans are able to recognize geometric similarities on parts of objects, like the windows on a building facade. The latter usually cannot be mapped to each other by an algebraically closed set of transformations (see Figure 2.2).

2.2 Partial Symmetries

While the classical type of symmetries (we refer to them as global) can be observed in a variety of shapes, these require transformations that map the entire object to itself. This work is focused on **partial symmetries**, i.e. self-similarities that occur on parts of a shape. These types of correspondences occur more often than global symmetries and allow for decomposing (segmenting) shapes into recurring pieces, which can be used to analyze or further process a shape.

[1] Groups containing translation are infinite. Therefore, not all elements can be represented physically.

Symmetry detection and analysis become more complicated when only unknown parts of the input object correspond to each other. In addition to finding the inducing transformations, the shape must be segmented into pieces. In Figure 2.2, the object to the right has three global symmetries (mirroring along the xy, xz, and yz plane) but also consists of 5 identical boxes that correspond to each other via horizontal translations. This example illustrates an additional difficulty of partial vs global symmetry detection: the transformations inducing partial symmetries do not always form closed algebraic structures as is the case with global self-symmetries.

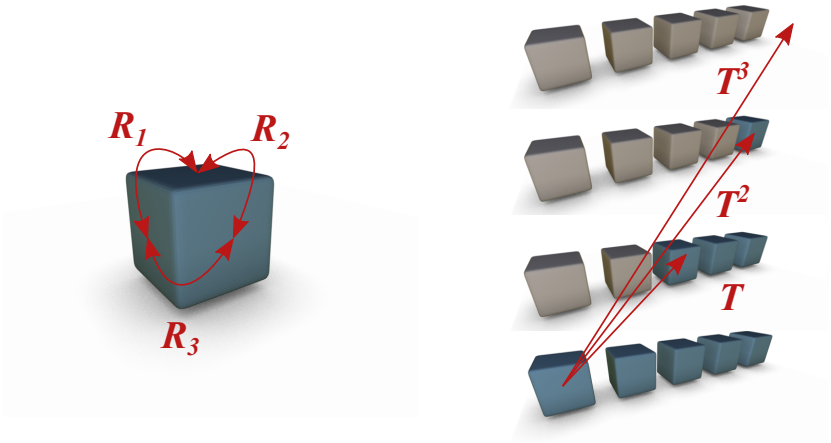


Figure 2.2: *The set of transformations inducing global symmetries form a closed group (left). This is not the case with partial symmetries (right). In the right example $T^3 = T{\cdot}T{\cdot}T$ does not produce a partial match.*

For global symmetries the rotations and reflection planes are all centered at the object center, and in general, the centroid of the object is a fix-point for any symmetry transformations. This is not the case for partial correspondences, which makes it more difficult to compute their inducing transformations.

In this work we investigate how partial symmetries characterize shapes and shape collections. We develop an abstract framework for shape decomposition into symmetric parts, which we call microtiles. We also look into how the microtiles induced by a certain type of rigid self-correspondences can be used for inverse modeling of shape variations. We show that these particular microtiles can be used to assemble a whole family of locally similar shapes.

2.3 Approximate Symmetries

When do we say that two geometric objects have symmetric or identical parts? Ideally, the pair of objects $A, B \subset \mathbb{R}^3$ must match exactly, because in Section 2.1 we defined: $A = B$ if and only if $\forall p \in A, \exists q \in B : q = p$. However, in practice this requirement is too strict and well behaved shapes that allow precise matching are rare. In addition, humans intuitively recognize "almost" identical shapes or "close" matches. Note that in many of these cases our cognitive ability

is not entirely based on experience rather than precise estimation of the geometry of the shape. We can, for example, recognize similarities between tools or weapons, despite deformations caused by aging, rusting, or defects. At the same time, recognizing that a pair of line segments have orientations that differ by less than 10°, but are not parallel to each other is not a skill an average human needs to posses. In this thesis, we are mainly interested in exact geometric correspondences and leave for future work the adaption of the proposed methods to approximate symmetry.

Even though our theoretical framework is focused on exact symmetries, practice demands a more general and robust approach. To address shape deformations caused by noise, numerical imprecision or modeling (artist) mistakes, we need to allow to match shapes approximately. One way to do this is to consider a distance function $d(\cdot,\cdot) : \mathcal{P}\left(\mathbb{R}^3\right) \times \mathcal{P}\left(\mathbb{R}^3\right) \to \mathbb{R}$ that measures pairwise similarity of geometric shapes ($\mathcal{P}(\cdot)$ is the power set of $\cdot$). We can choose a threshold t and identify two shapes A and B if $d(A,B) < t$. Note that in the case of self-correspondences B is usually a transformed copy of A.

There is more than one way to define a distance measure, and we will define one such function in Section 4.4.2. Other measures include the Hausdorff distance

$$d_H(A,B) := \max\left\{\sup_{x\in A}\inf_{y\in B}||x-y||_2, \sup_{y\in B}\inf_{x\in A}||x-y||_2\right\}$$

or integrating point-wise square distances: For $B = T(A)$ with T a transformation

$$d_2(A,T(A)) := \int_{x\in A} ||Tx - p_M(Tx)||^2 dx$$

where $p_M(y)$ is the projection (or closest point) of y onto M.

Note that, typically, defining approximate symmetries with a distance function does not preserve the group structure of the inducing transformations, even for global symmetries. It is usually possible to have $d(A,T_1(A)) < t$ and $d(A,T_2(A)) < t$ while $d(A,T_1T_2(A)) > t$. Hence, symmetries defined this way are not necessarily transitive, making them harder to detect.

Another important aspect of approximate symmetry detection is the error tolerance, usually achieved by the threshold t. It is usually user defined and application dependent.

In this work we consider well-behaved shapes with no noise and minimal deformations. Nevertheless, we derive a detection method that handles approximate symmetries and deal with numerical imprecision as well as some triangle mesh inconsistencies typical for real world 3D models modeled by humans. In addition, we demonstrate that our detection method can be extended to handle small shape deformations caused, for example, by artist mistakes. However, we leave for future work the application to more challenging scenarios such as scanned models subject to noise, missing parts, and other artifacts. We therefore use a strict threshold on our distance metric.

3 | Building Blocks from Correspondences

In this chapter, we develop a theoretical framework for characterizing collections of shapes by elementary pieces we refer to as building blocks. We introduce a formal model for shape segmentation into "microtiles" - elementary pieces induced by a set of input correspondences that define a point-wise equivalence relation between surface points. Each mapping matches parts of the input model. We convert the initial set of arbitrary overlapping surface patches into a unique set of microtiles and show that the latter encode all partial symmetries of the input shape.

We then consider a specific correspondence model called r-similarity and prove that the microtiles of a given shape $\mathcal{S}$ characterize a whole space of shapes similar to $\mathcal{S}$ (up to zero-area deviations). This insight is the major theoretical contribution of the thesis and is important for various applications, including Inverse Procedural Modeling (see Section 3.5), where the microtiles can be used as a tiling grammar - a set of building blocks and assembly rules that allow for the generation of a vast set of shape variants.

The contributions of this chapter are discussed in [KBW+12]. This chapter consists of two main parts. First, we develop a model for representing partial symmetries with microtiles. We consider a shape $\mathcal{S}$ and set of mappings $\mathcal{F}$ that maps subsets of $\mathcal{S}$ back to $\mathcal{S}$, in a way that it defines an equivalence relation among the points of $\mathcal{S}$. Usually, the regions matched by the pairwise relations overlap arbitrarily. This provides a notion of redundancy, which we then convert into canonical building blocks.

We formally prove a number of interesting properties of the microtiles: First, the construction is unique and canonical: It does not require any choices or parameters in addition to the input correspondences. Furthermore, microtiles are disjoint and different types of tiles do not have partial correspondences among each other or themselves, meaning that if parts of two microtiles match, then the entire pieces match as well. Finally, unions of microtiles have the intersection of their associated cliques of transformations as permissible operations. Thus, we can derive all partial symmetry properties of a shape from our decomposition by simple set operations.

In the second part of the chapter, we apply the abstract model for a specific type of point-wise correspondences. We consider **r-similarity** – the matching of rigid neighborhoods with radius r around points on the input surface. This yields a point-wise equivalence relation, from which we can build r-microtiles. Furthermore, a shape $\mathcal{S}_2$ is considered r-similar to $\mathcal{S}_1$ if all points $\mathbf{y} \in \mathcal{S}_2$ are r-similar to some point $\mathbf{x} \in \mathcal{S}_1$. We can now characterize the space of r-similar shapes: We prove that all shapes $\mathcal{S}_2$ that are r-similar to $\mathcal{S}_1$ can be constructed

by rigidly assembling r-microtiles of $\mathcal{S}_1$ up to a set of points with zero area. The assembly is disjoint and unique.

3.1 Related Work

Early approaches to structuring partial symmetries by building blocks used simple region growing [MGP06, BBW+09b], which does not lead to canonical results because they depend on the initialization. It is also not clear how to reassemble such tilings to form new shapes. Another way of structuring pairwise correspondences is to look for algebraic regularity in the domain of the transformations, for example by detecting commutative grids [PMW+08, MBB10]. This describes the structure of the correspondences only partially, as the input contains only excerpts of the symmetry groups (which are usually infinite), and irregularly placed instances are not captured. In other words, an algebraic regularity is usually not present in the domain of transformations, as translational components cannot be captured inside a finite group. Our decomposition gives closed groups when viewed as exchangeable parts, i.e., as permutations of sets, but not in the transformation domain.

An alternative is to enumerate the overlapping regions of pairwise matches [BWS10]. This implicitly encodes all symmetry information, but does not expose its structure. Based on this, Bokeloh et al. [BWS10] use the boundaries of partial r-symmetry in order to cut out **dockers** that form a context-free shape grammar. It encodes a set of objects that are r-similar to the input exemplar. Due to the restriction to context-free grammars, their method must avoid intersections of cuts. Our approach is the opposite, performing all cuts, even for continuous symmetries (which their method explicitly excludes). Thereby, we can span the complete space of r-similar objects with assemblies of microtiles, while context-free tiles only cover a restricted subset. Lifting this restriction and understanding the set of r-similar models was the main motivation of our work.

We should note that local similarity is often used in texture synthesis [EL99, WL00] and their analogs in geometry synthesis [BIT04, ZHW+06, Mer07] but these methods use variational formulations that do not provide insight into the structure of the shape space. In contrast, our approach explicitly provides building blocks and rules that span the shape space but restricts us to exact r-similarity.

In terms of representing partial symmetry, our method is related to the work of Lipman et al. [LCDF10]: Their method also starts from point-wise equivalence relations and computes cliques in correspondence graphs. Unlike our approach, they use a spectral clustering technique that is robust even under noisy and ambiguous data. The result is a symmetry factored embedding that maps points with a similar symmetry structure to nearby points in a Euclidean space. A subsequent clustering in this space produces a partitioning that resembles our microtiles. However, our model for dealing with partiality is different. Rather than weighting the percentage of mapped points, we model the functions and domains of partial matches explicitly. This allows us to formally study the re-

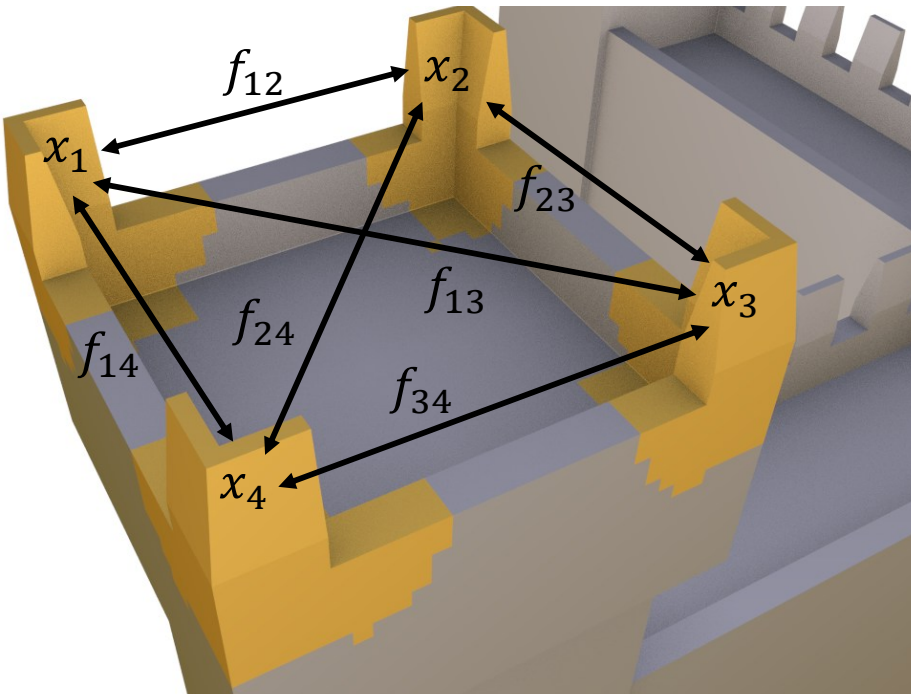


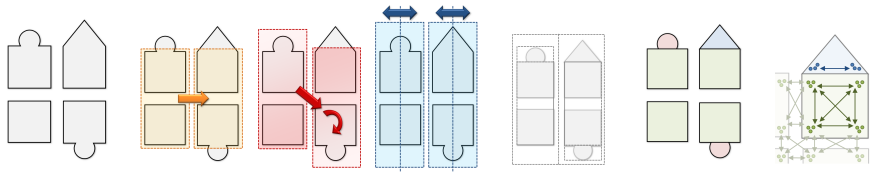
Figure 3.1: *A graph representation of an example set of partial mappings. The nodes are (colored) points on the surface and the edges are mappings. As illustrated, all connected components of the graph have to be cliques by definition.*

sulting tile decompositions and gives us a formal framework for how general mapping functions should be treated. In this chapter, we provide complementary insights: We focus on the study of the resulting tiles and their properties, and study how the class of shapes can be assembled out of those tiles. In addition, the similarity model in [LCDF10] is partly based on diffusion, making it difficult to determine the exact boundary of the corresponding region. This makes it unsuitable for the applications we discuss in the subsequent chapters of this thesis.

3.2 Correspondences and Microtiles

We consider a set of functions that relate points on a geometric shape, and decompose it into classes of points that agree to be mapped simultaneously to other points (s. Figure 3.1). In other words, we group the subsets of surface points that are always mapped together to their similar counterparts.The result is a set of minimal tiles that cannot be split further into pieces with differing (symmetry) properties. The only requirement on the mappings is that, as a set, they form an equivalence relation.

Definition 1. Input: In the following, we use $\mathcal{S} \subset \mathbb{R}^3$ to denote the exemplar piece of geometry for which we want to compute a microtile decomposition. Let



(a) (b) (c) (d) (e)

Figure 3.2: *Construction of microtiles: (a) Input geometry $\mathcal{S}$. (b,c,d) We consider partial correspondences $\mathcal{F}$ within $\mathcal{S}$. (c) The geometry is cut at the boundaries of the partial matches. (d) This yields the microtile decomposition. (e) Each microtile is characterized by cliques of equivalent points; cliques with the same set of transformations form the same class of microtiles.*

$$\mathcal{F} \subseteq \{(\mathcal{P}, f) | \mathcal{P} \subseteq \mathcal{S}, f : \mathcal{P} \to \mathcal{S}, f \text{ is a homeomorphism}\} \tag{3.1}$$

be a set of functions f that map subsets $\mathcal{P} \subset \mathcal{S}$ of the exemplar back to itself, in a topology preserving way. In other words, $\mathcal{F}$ is a set of partial correspondences on $\mathcal{S}$. The sets $\mathcal{P}$ identify the parts of $\mathcal{S}$ the functions f act upon.

Definition 2. Equivalence of points: Given a set of mappings $\mathcal{F}$, we can now say that two points $\mathbf{x}, \mathbf{y} \in \mathcal{S}$ are equivalent or similar, if there is a mapping $f \in \mathcal{F}$, which maps $\mathbf{x}$ to $\mathbf{y}$:

$$\mathbf{x} \equiv_f \mathbf{y} \quad :\Leftrightarrow \quad \exists (\mathcal{P}, f) \in \mathcal{F} : \mathbf{x} \in \mathcal{P} \text{ and } f(\mathbf{x}) = \mathbf{y} \tag{3.2}$$

We require that this relation be an equivalence relation. This means, we must choose $\mathcal{F}$ in a way such that this induced point-wise relation is:

- Reflexive: $(\mathcal{S}, id) \in \mathcal{F}$
- Symmetric: $\mathbf{x} \equiv_f \mathbf{y} \Rightarrow \mathbf{y} \equiv_g \mathbf{x}$ for some $g \in \mathcal{F}$
- Transitive: $\mathbf{x} \equiv_f \mathbf{y}$ and $\mathbf{y} \equiv_g \mathbf{z} \Rightarrow \mathbf{x} \equiv_h \mathbf{z}$ for some $h \in \mathcal{F}$

This is illustrated in Figure 3.1 where points on a surface are mapped based on partial symmetries. Having an equivalence relation of such mapping means that if we construct a graph with the surface points as nodes and the mappings between them as edges, then each connected component of this graphs will be a clique. The vertices in each clique form an equivalence class with respect to $\mathcal{F}$.

The equivalence classes given by the set of mappings already provide a decomposition of the input into sets of points. However, there are usually infinitely many such classes (or cliques), which makes this kind of decomposition impractical. In the following, we therefore cluster cliques with the same set of mappings.

Definition 3. Pointwise correspondence sets: For any $\mathbf{x} \in \mathcal{S}$, let $F_o(\mathbf{x})$ denote the set of all outgoing pointwise correspondences:

$$F_o(\mathbf{x}) = \{f | (\mathcal{P}, f) \in \mathcal{F}, \mathbf{x} \in \mathcal{P}\} \tag{3.3}$$

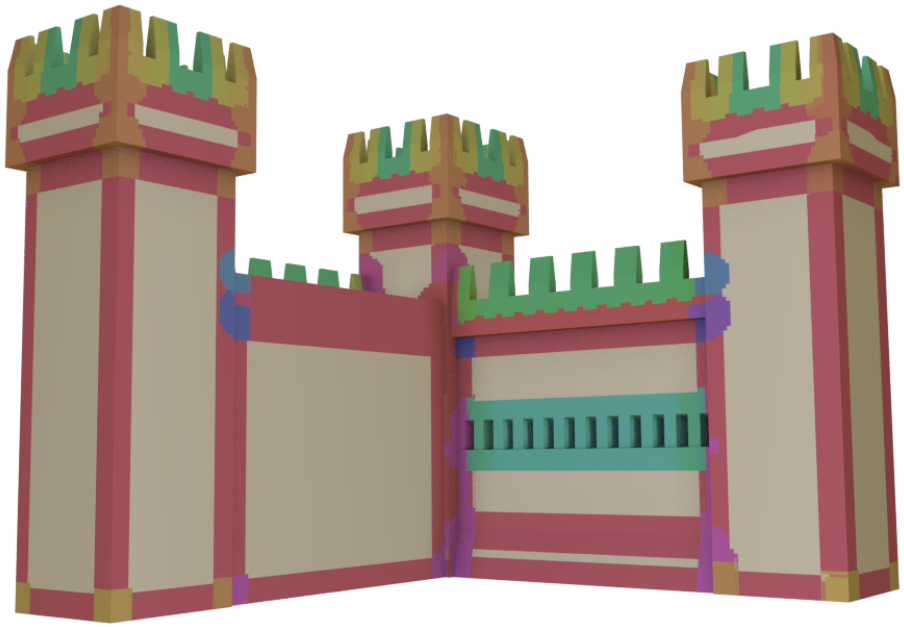


Figure 3.3: *A castle wing decomposed into microtiles w.r.t. r-symmetry. The colors indicate the class of the tiles.*

Definition 4. Instances of Microtiles: Comparing $F_o(\mathbf{x})$ for each point $\mathbf{x}$ introduces a new equivalence relation on $\mathcal{S}$: A maximal connected component of points $\mathbf{x}$ that have the same set $F_o(\mathbf{x})$ forms the same **instance** of a **microtile**. We require connected components at this point because, as we show later, these form the building blocks required for constructing new shapes.

Definition 5. Classes of Microtiles: Two instances of microtiles τ_1, τ_2 are from the same **class** if and only if they are matched by a correspondences $f \in \mathcal{F}$. This is because if two instances share a single pointwise correspondence, the whole instances must already be mapped by a correspondence from $\mathcal{F}$, by the following arguments: First, by definition, the same set of transformation applies to all points in a tile, implying that all points are mapped simultaneously. Second, any transformation is required to be a homeomorphism. Thus, connected components are preserved. In the following, we will denote the microtile decomposition of a shape $\mathcal{S}$ by $\mu(\mathcal{S})$. Examples of microtiles are visualized in Figures 3.3 and 3.1. Microtiles are colored by their classes.

Lemma 1. Elementary properties: There are several important properties of microtiles: First of all, for any $\mathcal{S}$ there **exists a unique** decomposition $\mu(\mathcal{S})$. This follows directly from the definition. In addition, if a point $\mathbf{x}$ is not equivalent to any other points, it belongs to a microtile characterized by the set $F_o(\mathbf{x})$ containing only the identity mapping.

For any point $\mathbf{x} \in \mathcal{S}$ there is **exactly one microtile** $\tau \in \mu(\mathcal{S})$ that covers $\mathbf{x}$ in $\mathcal{S}$. This is also straight-forward: for each point $\mathbf{x} \in \mathcal{S}$ there is exactly one set of mappings to equivalent points. Thus there is exactly one microtile that contains $\mathbf{x}$. Thus, microtiles form a **disjoint** partition of $\mathcal{S}$.

Finally, a microtile can be globally mapped to itself, but **not partially**. If a point on the tile can be mapped to another one by f, then f maps all other points on the same tile as well, because otherwise, we would have introduced varying cliques of mappings within the same tile, contradicting the definition.

Lemma 2. Cutting: We can define instances of microtiles in an alternative way: An instance τ of a microtile is a maximal, connected set such that:

$$\forall \mathbf{x_1}, \mathbf{x_2} \in \tau : \forall (\mathcal{P}, f) \in \mathcal{F} : (\mathbf{x_1} \in \mathcal{P} \Leftrightarrow \mathbf{x_2} \in \mathcal{P})$$

The equivalence to the definition above is straightforward. This formulation shows that we can characterize the tile instances by the intersection of the domains of all partial mappings. Intuitively, we can think of **cutting** at the boundaries of all partial matches.

Lemma 3. Combinations of tiles: Let $Q \subseteq \mu(\mathcal{S})$ be a set of tile instances. We then consider a union of tiles $\bigcup_{\tau \in Q} \tau$. We now want to find the set of symmetry transformations of Q, i.e., the set of $f \in \mathcal{F}$ that map all tile instances in Q simultaneously back to $\mathcal{S}$. From the definition, this is obviously $\bigcap_{\tau \in Q} F_o(\tau)$, i.e., the intersection of all mapping functions associated with the individual tiles. This also holds if Q contains arbitrary fragments of tiles: Otherwise, we would have partial symmetries of tiles. Hence, the microtile decomposition encodes all partial symmetries of $\mathcal{S}$ and they can be computed easily and efficiently by set operations.

Lemma 4. Permutation groups: Microtiles also have an algebraic regularity model: If we restrict any pair of mappings $\{f, f^{-1}\} \in \mathcal{F}$ that acts on a tile τ to the domain of this tile instance, we obtain an operation that swaps two tiles $\tau, f(\tau)$, not affecting the shape of $\mathcal{S}$. The set of all such permutations for all microtiles generates a group of permutations that characterize the symmetries of the shape. Please note, that neither the transformations involved nor the input set $\mathcal{F}$ has a group structure in general. By definition, the permutation group of each tile class is maximal with respect to sets of possible permutations, and the union of tiles of the same class are the maximal subset of $\mathcal{S}$ with that property.

3.3 Microtiles for r-Symmetry

The above definition of microtiles is still abstract; whether the concept of microtiles is useful or not depends on the input set of mappings that determine the building blocks. In order to demonstrate a concrete application, we derive some of the properties for a microtile decomposition based on partial r-symmetry as defined by Bokeloh et al. [BWS10].

Definition 6. r-Similarity and r-Symmetry: From now on, let $\mathcal{T} = E(3)$ be the group of rigid transformations of $\mathbb{R}^3$. Let $N_r(\mathbf{x})$ be the spherical r-neighborhood of $\mathbf{x}$ in $\mathcal{S}$ with respect to Euclidean distance. We first define a local notion of equivalence by matching neighborhoods of points: The **points**

$\mathbf{x}$ and $\mathbf{y}$ are r-similar under a transformation $\mathbf{T} \in \mathcal{T}$ if and only if their local neighborhoods match under a rigid motion. We denote this by:

$$\mathbf{x} \overset{r,\mathbf{T}}{\leftrightarrow} \mathbf{y} \quad :\Leftrightarrow \quad \mathbf{T}(N_r(\mathbf{x})) = N_r(\mathbf{y})$$

r-Symmetry: This relation, which we call **r-symmetry**, is an equivalence relation because $E(3)$ forms a group and its actions are isometries with respect to Euclidean distance, i.e., they do not change r. In the following, let $\mathcal{F}$ denote the partial mappings induced by the set of all pairwise r-symmetry relations.

r-Similarity: We can use the same matching model to compare shapes: Let $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{S}$ be two shapes in $\mathcal{S}$, with distance larger than r, and $\mathbf{x} \in \mathcal{S}_1, \mathbf{y} \in \mathcal{S}_2$. We call $\mathcal{S}_2$ r-similar to $\mathcal{S}_1$ if every point in $\mathcal{S}_2$ is equivalent to at least one (arbitrary) point in $\mathcal{S}_1$. Formally:

$$\mathcal{S}_2 \text{ is } r\text{-similar to } \mathcal{S}_1 :\Leftrightarrow \forall \mathbf{y} \in \mathcal{S}_2 : \exists \mathbf{x} \in \mathcal{S}_1 : \mathbf{x} \overset{r,\mathbf{T}}{\leftrightarrow} \mathbf{y}.$$

Obviously, r-similarity is reflexive and transitive but in general not symmetric.

In the following, we will consider decomposition of surfaces (e.g. triangle meshes) into microtiles with respect to r-symmetry, and we will show a number of interesting properties that these particular microtiles have. The main theoretical result in this thesis is the proof that these microtiles are a canonical description of the space of r-similar shapes to the input shape. Before we continue, we will make further restrictions to the input in order to make the further model well defined:

- $\mathcal{S}$ is bounded.
- $\mathcal{S}$ is a 2-manifold (with or without boundary).
- $\mathcal{S}$ is piecewise smooth; we assume a union of a finite number of facets, each of which is a an algebraic surface bounded by a finite set of algebraic curves.

The last point allows representations like meshes of trimmed NURBS, which can represent (exact) rotational symmetries. Our current practical implementation is actually based on triangle meshes, where exact matching rules out these symmetries.

Types of Microtiles

When constructing the microtiles according to r-symmetry, one can distinguish between two types of microtiles – those with finitely and those with infinitely many equivalent counterparts. The restrictions on the input surface to a finite mesh of polygons or patches allow us to simplify the cases where infinitely many

transformations map a point to an r-symmetric one. If a tile is characterized by infinitely many transformations, then these can be represented by a continuous function parameterizable in one or two dimensions. This kind of symmetries (here r-symmetries) are analyzed by Gelfand and Guibas [GG04] and the symmetric points are called *slippable*. Thus we have two types of tiles with respect to the size of their defining transformation sets:

- **Discrete pieces**: A single piece of geometry that is instantiated once or more by a (finite) discrete set of symmetry transformations.
- **Slippable pieces**: These pieces (and their r-neighborhood in their instantiation) are slippable. They can be planar, spherical, cylindrical, helical, surfaces of revolution, or extrusions, e.g. edges (see Gelfand and Guibas [GG04]).
  - Planar, spherical, cylindrical microtiles: These form area elements, represented by a single point and its slippage properties. They can be expanded to arbitrary area covering the underlying primitive, but need to be bounded by other tiles (or self-closed, e.g. for spheres)
  - Helical tiles, extrusions, surfaces of revolution: These can be expanded in one dimension, forming curve primitives. They are either self-closed (surfaces of revolution) or bounded by other tiles (helical tiles, edges).

In Figure 3.3 the gray-colored points are all instances of a single class of microtiles because their r-neighborhoods are planar. We call them *2-slippable* because it is possible to parameterize their set of r-symmetry transformations in 2 dimensions. Analogously the yellow-colored points correspond to multiple *1-slippable* classes of microtiles. For triangle meshes, only planar 2-slippable elements and 1-slippable extrusions are possible.

Complexity: The set of microtiles can be encoded with a finite number of real parameters. For a triangle mesh, this is easy to see. A single triangle is always described by a finite set of different microtile classes. Inductively, adding a triangle to a set of finite classes of tiles can only create a finite number of new such classes, bounded by a finite graph of straight edges. We conjecture that a similar property holds for a mesh of general algebraic surfaces as well, but a formal proof is beyond the scope of this document.

3.4 The Space of r-Similar Shapes

In this section, we discuss how the microtiles of an exemplar $\mathcal{S}$ characterize the space of all shapes that are r-similar to $\mathcal{S}$. We discuss two different aspects: First, we show that all such shapes can be assembled from a disjoint union of microtiles, and that this decomposition is unique (up to global symmetries of the tiles, which has no effect). Second, we give some necessary conditions for a shape grammar that constrains how the tiles can possibly be fit together.

Theorem 1. The Space of r-Similar Shapes: The main result that we want to show is the following: Let $\mathcal{S}_1, \mathcal{S}_2$ be valid input surfaces in the sense of our definition. Furthermore, let $\mathcal{S}_2$ be $(r+\epsilon)$-similar to $\mathcal{S}_1$ for $\epsilon > 0$. Then $\mathcal{S}_2$ can be constructed (completely covered) by a disjoint union of r-microtiles from $\mu(\mathcal{S}_1)$, using only transformations from $\mathcal{T}$ to arrange the pieces.

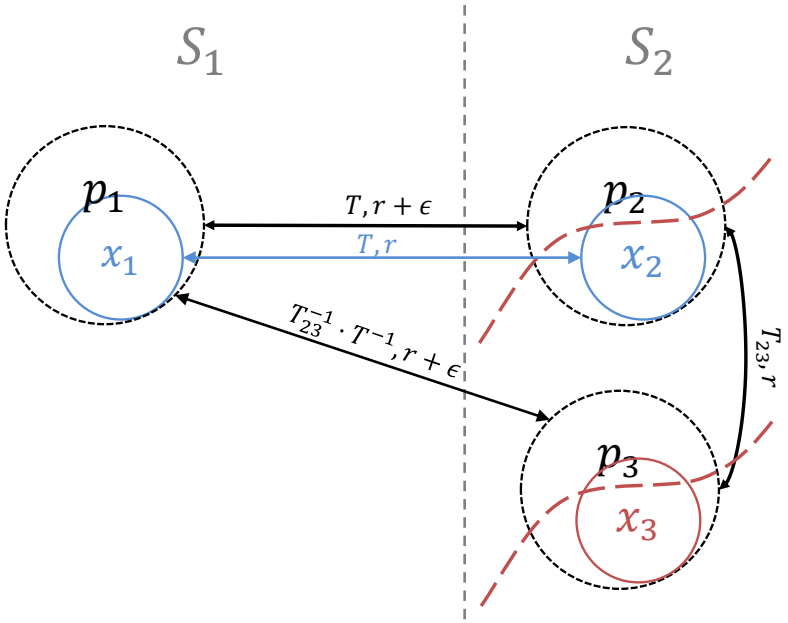


Figure 3.4: *Representation of the points in the proof. The $r+\epsilon$ symmetries and neighborhoods are marked with black arrows and circles, r-symmetries and neighborhoods with blue. We show that the red tile border can not cross the image of an r-microtile on $\mathcal{S}_2$.*

Proof: Because $\mathcal{S}_2$ is $(r+\epsilon)$-similar to $\mathcal{S}_1$, we can find a correspondence for each point of $\mathcal{S}_2$ and the point-wise equivalence classes or cliques of $\mathcal{S}_2$ are a subset of those of $\mathcal{S}_1$.

It remains to show that points on the same tile τ of $\mu(\mathcal{S}_1)$ cannot be r-symmetric to points on different tiles of $\mathcal{S}_2$. Assume that this is not the case and the points on a tile of $\mathcal{S}_2$ can only be mapped to points of at least two tiles of $\mathcal{S}_2$, i.e. in $\mathcal{S}_2$ exists a tile boundary not observed in $\mathcal{S}_1$ (Figure 3.4, red dashed line). Consider two points $\mathbf{p}_1, \mathbf{x}_1 \in \tau \in \mu(\mathcal{S}_1)$ and a transformation $\mathbf{T} \in \mathcal{T}$ such that $dist(\mathbf{p}_1, \mathbf{x}_1) < \epsilon$, and $\mathbf{p}_2 := \mathbf{T}(\mathbf{p}_1)$ and $\mathbf{x}_2 := \mathbf{T}(\mathbf{x}_1)$ are not in the same r-tile of $\mathcal{S}_2$ (s. Figure 3.4). In other words, we are interested in a pair of points $(\mathbf{p}_2, \mathbf{x}_2 \in \mathcal{S}_2)$ on different tiles, but within the ϵ-neighborhood of a tile boundary that is only present in $\mathcal{S}_2$. We will show that $\mathbf{p}_2, \mathbf{x}_2$ cannot belong to different tile classes, contradicting the assumption of the existence of an additional tile boundary in $\mathcal{S}_2$.

Because, by assumption $\mathbf{p}_2, \mathbf{x}_2$ do not belong to the same microtile class, there must exist $\mathbf{p}_3, \mathbf{x}_3 \in \mathcal{S}_2$ with $\mathbf{p}_2 \overset{r,\mathbf{T}_{23}}{\leftrightarrow} \mathbf{p}_3$ such that $\mathbf{x}_3 := \mathbf{T}_{23}(\mathbf{x}_2)$ is not r-symmetric to $\mathbf{x}_2$ (Figure 3.4). From the theorem statement (the condition on

$r+\epsilon$ similarity of $\mathcal{S}_1$ and $\mathcal{S}_2$), having $\mathbf{p}_1 \overset{r+\epsilon,\mathbf{T}}{\leftrightarrow} \mathbf{p}_2$ and $\mathbf{p}_2 \overset{r,\mathbf{T}_{23}}{\leftrightarrow} \mathbf{p}_3$ implies $\mathbf{p}_1 \overset{r+\epsilon,\mathbf{T}\cdot\mathbf{T}_{23}}{\leftrightarrow} \mathbf{p}_3$. Therefore, $\mathbf{x}_1 \overset{r,\mathbf{T}\cdot\mathbf{T}_{23}}{\leftrightarrow} \mathbf{x}_3$. The latter holds because if $\mathbf{p}_2$ and $\mathbf{x}_2$ are closer than ϵ, the r-neighborhood of $\mathbf{x}_2$ is completely inside the $(r+\epsilon)$-neighborhood of $\mathbf{p}_2$. The same holds for $(\mathbf{p}_1, \mathbf{x}_1)$ and $(\mathbf{p}_3, \mathbf{x}_3)$ respectively. However, $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ being r-symmetric to each other contradicts that the latter two belong to different r-microtiles of $\mathcal{S}_2$.

□

The theorem shows that out of the r-microtiles it is possible to construct any shape r-similar to a given exemplar up to a subset of measure zero. This holds because it is possible to choose ϵ arbitrarily small. One can think that this result is actually stronger, i.e. that one can let ϵ converge to 0 and argue that the statement can be transferred to the limit, but we were not able to show that this is always possible. Observe that for the theorem to hold in the limit case, the properties of the tiles like number, size, adjacency should also change continuously with r. In the case of triangle meshes for example this is always the case up to a countable amount of values where discontinuities in all of the mentioned properties occur. To illustrate this consider a single triangle. For large enough radius, the whole triangle will be a single microtile, while if the radius is small enough, the triangle will be decomposed into separate microtiles for corners, edges and a plane. For each value of r inside this range at which the number of microtiles changes, the set of r-similar shapes for $\epsilon > 0$ and $\epsilon = 0$ differs.

This limitation is not very important, because the values of r for which the microtiles do not suffice to characterize the space of shapes r-similar to an exemplar are finitely many and do not pose a real concern in practice.

3.5 Towards a Microtile Shape Grammar

In addition to having a unique decomposition into microtiles, we can constrain the construction of r-similar shapes further. We will now show that we cannot only learn the tiles from the exemplar, but also some rules of how they can possibly be assembled. To that end, we first have to define boundaries of a tile:

Let τ_1, τ_2 be two instances of microtiles of different type. The **boundary** b between τ_1, τ_2 is the set of points that have distance zero to both τ_1, τ_2, i.e., $b = \overline{\tau_1} \cap \overline{\tau_2}$, where the bar denotes set closure. Because we assume manifold input, and tiles are disjoint, the boundaries are (multi-)curves (possibly consisting of multiple fragments). Only two tiles can share a common curve except from isolated points that can be a boundary to more than two tiles.

We now show that tiles can only be assembled along boundaries that have been observed in the exemplar. We have the following theorem:

Theorem 2. Let $\mathcal{S}_2$ be $(r+\epsilon)$-similar to $\mathcal{S}_1$. Let τ_1, τ_2 be two different instances of r-microtiles of $\mathcal{S}_1$. Let $\tau_1' = \mathbf{T}_1(\tau_1), \tau_2' = \mathbf{T}_2(\tau_2), \mathbf{T}_1, \mathbf{T}_2 \in \mathcal{T}$ be instances

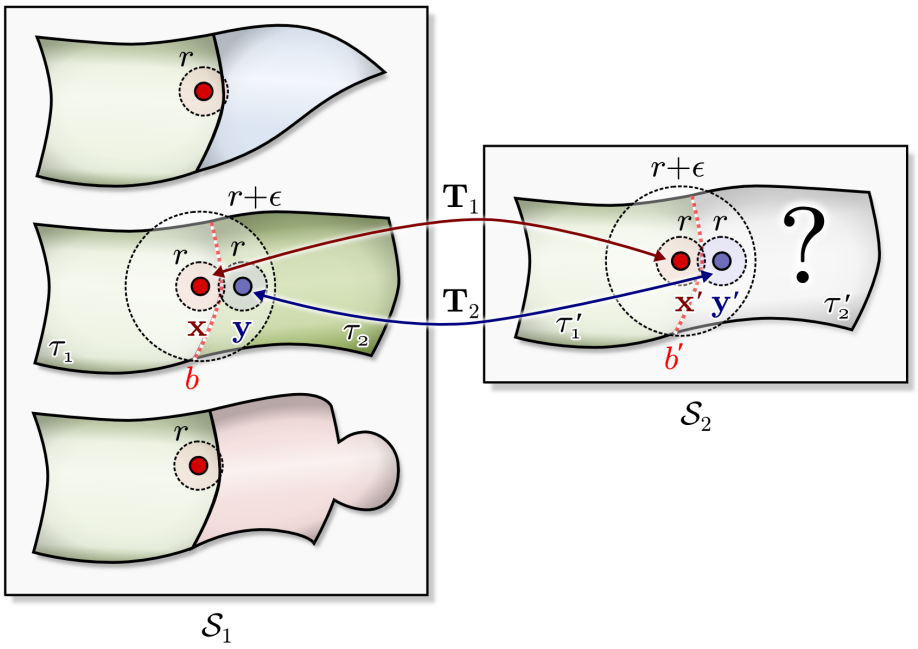


Figure 3.5: *Microtiles can only be assembled with pairwise adjacency relations as in the exemplar (left).*

of tiles of the same types as τ_1, τ_2 in $\mathcal{S}_2$, i.e., $\tau_1', \tau_2' \subset \mathcal{S}_2$. Let τ_1', τ_2' share a common boundary b'. Then, there must exist instances of τ_1, τ_2 in $\mathcal{S}_1$ that share a boundary b. Furthermore, we can always find a pair of instances $\tau_1, \tau_2 \in \mathcal{S}_1$ such that $\mathbf{T}_1 = \mathbf{T}_2$ and $b' = \mathbf{T}_1(b)$.

Proof. We show that $b \subset \mathcal{S}_1$ must exist (Fig. 3.5) with arguments similar to the proof of the previous theorem. We know that $\tau_1' \subset \mathcal{S}_2$ corresponds to one or more tiles in $\mathcal{S}_1$. Let $\mathbf{x}' \in \tau_1'$ and $\mathbf{y}' \in \tau_2'$ be two points, both with distance less than $\epsilon/2$ to the boundary b'. Because $\mathcal{S}_2$ is $(r+\epsilon)$-similar to $\mathcal{S}_1$, $\mathbf{x}'$ and its $(r+\epsilon)$-neighborhood map to a point $\mathbf{x} \in \mathcal{S}_1$ that is contained in one of these tiles. Let τ_1 denote this tile, and $\mathbf{T}_1^{-1} \in \mathcal{T}$ denote the mapping. Because of the $(r+\epsilon)$-similarity, $N_r(\mathbf{x}')$ and $N_r(\mathbf{y}')$ are both mapped back to $\mathcal{S}_1$ under $\mathbf{T}_1^{-1}$. As before, this implies that $\mathbf{T}_1^{-1}(\tau_2') \subset \mathcal{S}_1$ is also an instance of the microtile τ_2 (otherwise, we would have found a partial symmetry of τ_2). Therefore, we have constructed a pair of corresponding tiles with $\mathbf{T}_1 = \mathbf{T}_2$, which implies that the boundary $b = \mathbf{T}_1^{-1}(b')$ exists in $\mathcal{S}_1$. □

This shows that we can learn restrictions how to connect microtiles from the exemplar: We can collect all boundaries along which microtiles are neighboring in the exemplar $\mathcal{S}_1$ and allow only these combinations for building new shapes. For **discrete** microtiles, this is straightforward (we just need to enumerate the observed combinations). For **continuous** microtiles, the situation is more complicated: Continuous tiles can be neighboring to themselves. For example, the

tile of a plane consists of a single point only; finite pieces are formed by collections of points. This implies that continuous microtiles that have themselves as neighbors (which actually can be expected to be almost always the case) can be extended to form arbitrary kinematic surfaces of the type the tile corresponds to (planes, spheres, helical surfaces, etc). The only constraint is the boundary: The continuous tile must be bounded by observed boundaries to tiles of other types, which again can be continuous or discrete[1]. Any combinations of these boundaries are possible, as long as the adjacency of the boundary elements themselves is compatible as well. As an example, consider a flat wall with a single window in it that forms a single tile. This means, that in a newly constructed shape, we can insert an arbitrary number of windows into the plane, rotated and translated arbitrarily, as long as their distance is larger than r.

Discussion: The rules are necessary for assembling $(r+\epsilon)$-similar objects, but not sufficient. Following the (informal) arguments of Bokeloh et al. [BWS10], we conjecture that our rules of assembling microtiles according to previously observed "docking sites" are sufficient for creating r-similar objects, leaving only a gap of ϵ in the class of shapes described. However, a formal proof is subject to future work, as well as a constructive characterization of a shape grammar that can directly create new objects. Analogous to [BWS10], we call these assembly rules a **microtile grammar**. This grammar gives a more general and explicit version of the "shape matching grammar" in their paper, which is only defined implicitly. Both grammars are not context free, but rather resemble a jigsaw-puzzle. Our version provably covers all $(r + \epsilon)$-similar shapes. Please note that both are unrestricted, type-0 Chomsky grammars [Cho59] (in our case, including the special case of defining continuous pieces by their boundaries). They resemble a puzzle of pieces docked along their boundary curves rather than the hierarchical replacement scheme of traditional, context-free shape grammars.

3.6 Final Words

We have presented a formal model for decomposing a model into building blocks when we have correspondences that identify equivalent surface points. The basic idea is very simple: We group all connected points that are mapped by the same set of mapping functions. Nevertheless, the decomposition has a number of remarkable properties. As an abstract decomposition it is a unique and disjoint partition. It also encodes all consistent matches of subsets of the input surface. Applying it to the concrete correspondence model of rigid matches of local neighborhoods, we get the even stronger result that the tiles of the decomposition are sufficient to uniquely construct any model that is similar to the decomposed one in this sense.

In this document we demonstrate two main application areas where our approach is of utility. First, it provides a canonical representation of redundancy in

[1] Spherical mirotile types are an exception, because we can form a closed sphere, which is a bounded set that respects the neighborhood rule, i.e. having observed boundaries to other types of building blocks is not a requirement in this case.

shapes. With respect to given correspondences, we can consider the microtiles as elementary units. This has obvious applications in shape compression (see Chapter 6) and could be used to analyze collections of shapes for similarity (demonstrated briefly in Chapter 4). In addition to relating the tiles themselves, the relative arrangement of the tiles could probably, in future work, be used to characterize geometry independent of the actual geometry. A further application area is inverse procedural modeling, where we can characterize the space of shapes that is locally similar to an exemplar. In Chapter 7 we exploit this property to inversely decompose input models into manufacturing-ready constructor pieces that can build families of r-similar shapes.

4 | r-Symmetry Detection

In the previous chapter we introduced a theoretical model for shape decomposition into microtiles – building blocks derived by a set of correspondences that define an equivalence relation on the surface points of a given model. We also showed that for a specific correspondence functions (rigid r-neighborhood matching) the set of microtiles characterizes all shapes r-similar to a given exemplar. Now we address the problem of computing a microtile decomposition and show how to efficiently detect r-symmetric of points on triangle meshes. We first demonstrate that a microtile decomposition w.r.t. rigid r-symmetry is computable (see [KBW+12]). Afterwards, we introduce an efficient method for computing microtiles, which permits such involved analysis to be used in variety of geometry processing applications.

4.1 Introduction

Partial symmetry detection and its uses in the field of Computer Graphics has been studied extensively in the recent years (see Mitra et al. [MPWC12]). The advances in this area leave the impression that symmetry detection, and in particular detection of rigid partial symmetries is a well understood and broadly covered problem [MGP06, BBW+09b, LCDF10]. However, none of the proposed techniques is widely accepted to be robust and efficient enough to warrant its integration into existing geometric modeling software. Furthermore, the majority of the recent papers on this topic are focused on discovering **a subset** of the partial correspondences of an input model (or models), which suffices to allow further analysis of the data.

This makes it highly unlikely that these symmetry detection algorithms can be used to efficiently compute a microtile decomposition in the sense discussed in Section 3.3, where all point-wise equivalences must be computed in order to obtain the required decomposition. It might be possible to use some of the related techniques for partial symmetry detection to perform "brute force" search for possible correspondences. However, the amount of the potential and actual partial matches even for apparently simple models is so high, that the resulting running times of such implementation would make the microtile analysis impractical.

In this chapter, we present two novel algorithms for microtile decomposition and r-symmetry detection. First, we derive an implementation using the work by Bokeloh et al. [BWS10] as a starting point and use it to verify the theoretical concept from Chapter 3 and the proof of the main theorem in Section 3.4. The

first part of the chapter is discussed by Kalojanov et al. [KBW+12]. We then go on and derive an efficient and practical algorithm that is faster by two orders of magnitude compared to the first attempt and can be efficiently implemented on parallel architectures such as graphics processors. We are able to achieve decomposition times under a minute on a commodity PC for meshes of non-trivial complexity, detecting hundreds of thousands of partial correspondences .

4.2 Related Work

The algorithms for symmetry detection we discuss are closely related to the works of Bokeloh et al. [BBW+09b, BWS10]. There, the authors identify and match line features to detect similar or corresponding regions on the surface of the input model. Bokeloh et al. [BWS10] detect matching points that lie on pairs of r-symmetric cuts through the model. To decompose a shape into microtiles one needs to compute all such cuts. This implies that the matching algorithms used by Bokeloh et al. can be applied for microtiles, but are in general inefficient because of the larger number of candidate transformations that need to be evaluated.

Lipman et al. [LCDF10] propose a symmetry-aware distance metric and evaluation method based on spectral clustering. A common aspect of their method and our work is the grouping of surface points into equivalence classes, which they call orbits. Lipman et al. assume that the closer two points are located spatially, the more likely it is for them to have the same correspondences and therefore combine their similarity measure with diffusion. This makes their method robust to data with noise and deformations present in scanned models. We instead focus on clean surface models in order to compute the exact boundaries of each symmetric region, which is necessary for the applications in Chapter 6 and Chapter 7.

In addition to the closely related approaches we already mentioned, there are a number of recent methods for global and partial symmetry detection and extraction including [AMWW87, Ata85, BAK10, BBW+08, BBW+09a, BCBSG10, BWKS11, BWM+11, CK10, GCO06, GG04, KCD+03, KFR04, KLCF10, LTSW09, MBB10, MGP06, OSG08, PMW+08, PSG+06, RBBK07, RBBK10, RBB+10, SKS06, SOG09, TW05, XZT+09, ZPA95], all of which are systematically summarized in the state-of-the-art report by Mitra et al. [MPWC12].

4.3 Naïve Microtile Detection

We now describe an algorithm that computes the microtile decomposition of a manifold triangle mesh in polynomial time. This method is used by Kalojanov et al. in [KBW+12] as a first attempt at computing a complete microtile decomposition w.r.t. r-symmetry. We start with the abstract algorithm and then discuss its correctness and a concrete prototype implementation. Following Mitra et

al. [MPWC12], we proceed in three stages: **Feature selection, aggregation, and extraction**.

Feature Selection: We cannot test infinitely many transformations in the way they appear in slippable regions. Therefore, we first perform slippage analysis for all r-neighborhoods [GG04]. Afterwards, we ignore slippable regions in the remaining computation. Then, we compute *line features* [BBW$^+$09b]. For a triangle mesh, this is the subset of the edges with adjacent non-coplanar faces. We then generate *candidate transformations* by matching line features.

Aggregation: For each candidate transformation $\mathbf{T}$ and its inverse we match the whole scene $\mathcal{S}$ against $\mathbf{T}(\mathcal{S})$. An exact algorithm would compute an intersection of the two meshes (in practice, we will resort to an approximation, using voxels rather than the triangle meshes as representation [BWS10]). For each matching fragment, we record the matching element and the transformation indices in a table. After all transformations are processed the table encodes all detected partial r-symmetries for the shape.

Extraction: We extract a segmentation of the input scene into microtiles by region growing starting at an arbitrary (non-processed) element and expanding the current tile with elements that have the same set of symmetry transformations. We use the table we computed in the previous step to look up the transformations that map the geometry to r-symmetric parts of the surface. After the initial segmentation, we compute the equivalence classes of points (the cliques discussed in Figure 3.1). We transform the voxels that belong to a given microtile, and search in the overlapping voxels for the equivalent microtile instance.

4.3.1 Algorithm Overview:

1. We start by computing a voxel representation of the input geometry, that we store in an octree similar to Bokeloh et al. [BWS10].

2. We compute and mark all r-slippable voxels and treat them separately in the following steps.

3. We then detect the line features that indicate possible symmetry transformations s. [BBW$^+$09b]. We use them to compute the actual set of candidate transformations that map pairs of r-symmetric points to each other.

4. We compute the set of partial r-symmetries for the scene by iterating over the set of candidates and for each transformation T:

   - We mark all parts of the model r-symmetric w.r.t. $\mathbf{T}$ or $\mathbf{T}^{-1}$
   - We store the information in a global table.

5. We subdivide the scene into segments based on the symmetry transformation table we computed.

6. Finally, we find all equivalent tiles using the symmetry transformations that map voxels of each tile to r-similar voxels.

4.3.2 Correctness and Complexity

The above algorithm computes a correct microtile decomposition if there is no rigid transformation $\mathbf{T} \in \mathcal{T}$ such that points on different microtiles of the output are r-symmetric under $\mathbf{T}$. In other words we need to compute **all** transformations that can map two r-regions on $\mathcal{S}$ symmetrically. This is the major difference between our symmetry analysis and most of the related work [BWS10, BWKS11, LCDF10], where it suffices to find some, but not all, of the partial correspondences present in the model.

There are three cases of r-neighborhoods, that we need to consider in order to correctly decompose them in microtiles. These neighborhoods (as well as the respective microtiles) can be 2-slippable, 1-slippable or non-slippable. The 2-slippable surfaces on a triangle mesh can only be planes. They are characterized by a single microtile, and we check for its existence in the input model during slippage analysis.

If a point on a triangle mesh is 1-slippable, then its r neighborhood contains one or more edges, that are parallel to the line features of the input scene $\mathcal{S}$. Even though 1-slippable microtiles are mapped by infinitely many transformations to symmetric tiles, it is possible to consider segments of non-zero lengths instead of the infinitely small microtiles. Detecting discrete symmetries between such segments allows for decomposing the 1-slippable microtiles. To this end, we need to compute all transformations that map pairs of line features to each other (s. [BBW$^+$09b]).

It remains to find all valid transformations. Each of them maps a pair of non-slippable neighborhoods symmetrically. Observe that any such neighborhood has to contain at least two non-parallel edges (line features). Transformations that map pairs of non-parallel line features have to align the center of the shortest line segment between the two lines and the line directions. These transformations can be computed by exhaustively checking all possible pairs of features at distance no more than r from each other.

For n input triangles, the abstract algorithm performs no more than $\mathcal{O}(n^4)$ intersection computations of $\mathcal{S}$ against a transformed version of itself (which can trivially be computed in quadratic time). Each intersection test costs $\mathcal{O}(n^2)$ if performed brute force, and exhaustively testing all pairs of elements (triangles) for symmetries requires $\mathcal{O}(n^2)$ such tests.In practice, for non-degenerate scenes, $\mathcal{O}(n^2)$ such matches with slightly super-linear costs (using spatial data structures for intersection computation) could be expected, i.e. the total complexity is approximately $\mathcal{O}(n^3 \log n)$.

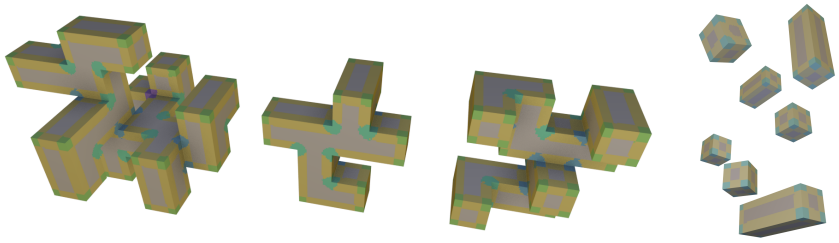


Figure 4.1: *Simple models. Left: Simultaneous decomposition of three simple meshes in one scene, $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$. Right: decomposition of 7 boxes in one scene. Equivalent tiles in each figure were computed and colored automatically. The number of different microtiles are 6, 5, and 5 for $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$. Each box on the right has 3 microtiles, as expected. The run-time for the naïve decomposition method of the first three meshes was around 10 min, the boxes on the right took approx. 1min.*

4.3.3 Results

We have implemented a simple prototype of the algorithm outlined above. We follow the method of [BWS10] and use a volumetric grid to discretize the symmetry information: cubes of side length h are annotated with transformations.

We have applied our prototype implementation to a few scenes to visualize the structure of the decomposition. For the tests we set the radius of symmetry to 0.008 (Figure 4.1) or 0.016 (all other tests) of the diagonal of the bounding box of the scene. The voxel size was set to $1/512$ of the diagonal ($1/256$ for Fig. 4.3). To prevent errors due to coarse discretization, classes of microtiles were computed only for microtiles larger than 32 voxels. Very small tiles usually indicate places where a finer discretization is required and we could not reliably compute the equivalence classes of such microtiles. Computing of the candidate transformations and the table that stores the set of transformations for each voxel are implemented in parallel. All test were performed on a single Intel Core 2 Quad Q9400 CPU with 4 cores running at 2.66GHz.

Figure 4.1 shows a very simple test scene composed out of boxes. The left hand side shows a scene of three different shapes, decomposed simultaneously. On the right, a simpler scene of independent boxes is decomposed. For these simple scenes, we obtain accurate results up to the resolution of the discretization. In Figure 4.2, we apply the algorithm to more complex meshes of architectural objects. We depict 2-slippable tiles in gray, 1-slippable in yellow (irrespective of the class), and only show the large tiles, as explained above. The corresponding unassigned area is shown in dark gray.

The computed decompositions are in most regions qualitatively correct, however, the grid-discretization leads to certain variations at the boundary. We observe some unassigned area, but its diameter is below r in all of the examples. Because the voxel-discretization does not permit a $1:1$ mapping, boundaries show some variability within voxel resolution (particularly visible at the sides of the courthouse).

Furthermore, rotational patterns are numerically problematic (e.g., overseg-

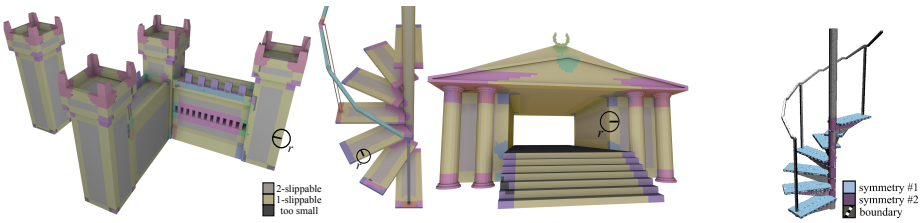


Figure 4.2: *More complex models: Castle (left), staircase (middle), courthouse (right) models decomposed into r-microtiles. The classes can be computed reliably only for microtiles larger than 32 voxels. The diameter of this uncertain area is smaller than r. Run-time: around 1 hour (castle), 18 min (staircase), and 40 min (courthouse). Far right: comparison to [BBW+09b].*

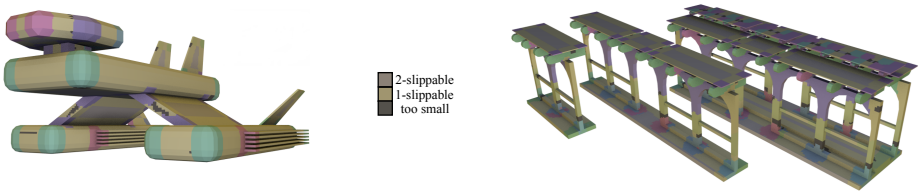


Figure 4.3: *A spaceship model (left) – the tiles are recognized correctly, but again, some unassigned area remains (gray). A cascade of models with increasing complexity (right) – the newly added parts create new tiles.*

mentation of the steps of the staircase). Similar results are obtained for the models in Figure 4.3. We compare our results to the previous method by Bokeloh et al. [BBW+09b], which is computationally mostly similar but uses (as most others) simple region growing for segmentation. Their method is similarly susceptible to discretization and boundary artifacts. It does not capture all symmetries (unlike ours), but samples prominent representatives due to the area/instance ratio heuristic employed. Global symmetries of the steps are detected, which do not affect the microtiles but are obtained implicitly with our new approach.

This implementation is only intended as a proof of concept, but there are already some direct applications: We can determine whether two shapes are r-similar, by matching their respective microtiles. The three box sculptures in Figure 4.1 are made of the same tiles, except from the leftmost, which contains one extra, unique tile, colored violet. Similarly, the isolated tower at the left of the castle in Figure 4.2 is r-similar to the castle, which contains additional tiles. A further example is demonstrated in Figure 4.3 (right). A sequence of models with increasing complexity is decomposed into microtiles, revealing the redundancy in the model collection.

4.3.4 Discussion

The voxel- and feature-based approach has some drawbacks we need to address: The precision of the decomposition is limited by the discretization of

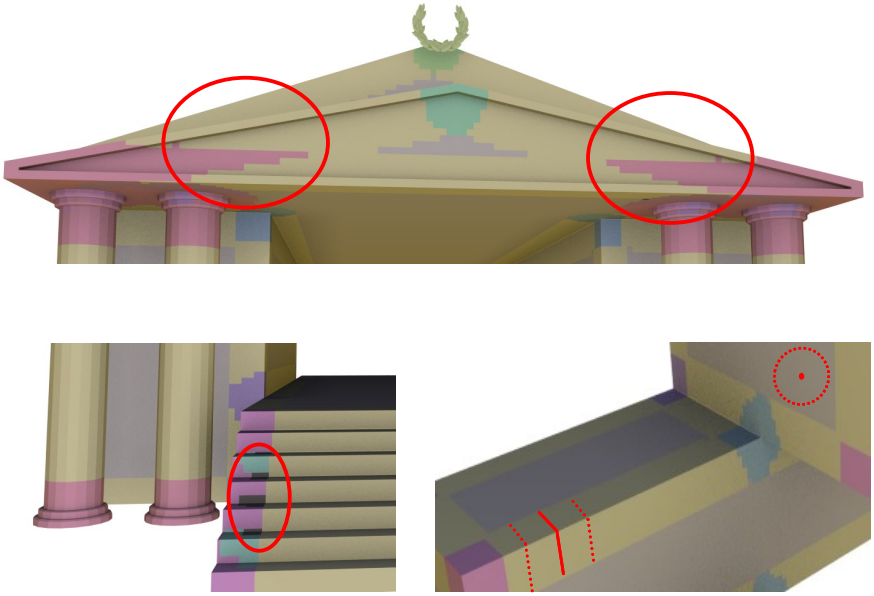


Figure 4.4: *Drawbacks of the naïve microtile extraction method: The global voxelization results in different voxel representations for equivalent microtiles (top) and artifacts inside voxels on tile boundaries (bottom left). One- and two-slippable microtiles are excluded from the analysis (bottom right).*

the scene. Because we voxelize the scene globally, equivalent microtiles will be decomposed in different ways, and will have a slightly different voxel representations (see Figure 4.4). We are sometimes unable to correctly compute the equivalence classes or cliques inside voxels on tile boundaries (see Figure 4.4). In order to reduce the artifacts on tile boundaries we post-process the initial segmentation based on pairwise matching. We transform the microtiles with each of the r-symmetry transformations to find their equivalent counterparts. Because a single tile will overlap multiple other parts, we gather votes from the overlapping voxels and select the tile that is most similar in terms of size and set of symmetry transformations.

Another practical issue is related to the precision with which we can compute the matrices for the actual transformations. Because we align each of them at a single corner feature, the mapping becomes more imprecise the longer the distance to the feature. In combination with the voxel quantization, the result is that near tile borders, r-symmetry detection becomes inconsistent and the set of transformations for many of the voxels is incomplete. This shows up as a large amount of small microtiles, that make further extraction of the equivalence classes virtually impossible. We address this problem by a filter: Near tile boundaries, we merge small tiles to neighboring larger one, whenever the voxel distribution and set of transformations of the smaller one suggest that it can be a part of the larger tile. We never discard a tile that has at least one

voxel completely surrounded by voxels on the same tile. This ensures that the area we filter will converge to zero if the voxel size does so.

The final important limitation is the runtime of the decomposition algorithm. It is rather large for two reasons. First, the algorithm performs all pairwise comparisons explicitly. Small test scenes compute in a few minutes, medium complexity scenes such as the castle require 1 hour (see Figure 4.1,4.2). Both the number of features and the required resolution for representing the symmetries are limiting factors, and both act quadratically on the run-time.

The second limiting factor for the runtime is the large number of self correspondences that we compute. Even geometrically simple models can exhibit a large number of partial self-symmetries, making a microtile decomposition expensive to compute in general. For example, the transformation candidates we test for the courthouse model in Figure 4.3 and Figure 4.4 is more than 3 million. To combat this, we try to discard candidate transformations as early as possible. We ignore transformations between corner features that fail to align all edges meeting at the corner. We discard duplicate transformations, and those that map corner points that are not r-symmetric. If a transformation matches two line features, we only consider it as a valid candidate if all 1-slippable voxels along the shorter feature a mapped to symmetric (1-slippable) voxels along the longer feature. Despite these optimizations, the run time and the number of remaining transformations remains rather large (see Figures 4.1, 4.2, 4.3).

4.4 Efficient r-Symmetry Detection

The microtile decomposition method used in [KBW+12] has several significant limitations. These fall into two categories: implementation specific and conceptual.

The algorithmic (or conceptual) challenges that were not solved come from the nature of the analysis required to obtain a microtile decomposition. We need to compute all rigid transformations that map any point on the input surface to another point such that their neighborhoods with radius r match. The number of such mappings is substantial regardless of the scene complexity, even without explicitly computing the infinite amount of matches on the slippable portions of the input model undergoing continuous r-symmetry.

On the other hand, implementation specific limitations are caused by the use of a global scene discretization to perform and record symmetry information about the object. In previous works [KBW+12, BWS10], the authors transform the complete scene and match it with itself using a spatial index structure (an octree) to compute and store self similarities. Apart from being inefficient, this approach leads to discretization artifacts, especially at the boundaries of the symmetric regions. These problems are amplified by imprecision caused by the floating point representation of the transformation matrices.

In the following, we will describe a more efficient and robust algorithm for r-symmetry detection and microtile decomposition. The new approach and its

implementation are faster by up to two orders of magnitude compared to the naive version. The significantly improved performance is essential, because it turns microtile detection for triangle meshes into a feasible pre-processing step for many geometry processing applications. Also, the method will allow for computing building blocks and cut the input surface exactly at the microtile boundaries, which is essential for computing 3D realizable building blocks in Chapter 7.

4.4.1 Feature-Based Discretization

We improve the performance the naïve microtile extraction algorithm in two aspects. On the one hand, we simplify and optimize the implementation of the individual steps required to obtain the building blocks, and we perform the most time-consuming parts in parallel. On the other hand, we propose a different approach in terms of algorithms and scene discretization. The latter makes the decomposition robust to discretization artifacts because unlike the naïve version, the segmentation no longer has to be performed on a per-voxel basis, which allows to reliably compute exact boundaries and microtile representations invariant under rigid transformations.

In the following, we will prove that in order to compute all equivalence classes of rigid r-neighborhoods on a triangle mesh it suffices to classify those centered at geometric corners or geometric edges. That is, we will be able to deduce a microtile decomposition for the entire input surface if we consider a finitely many feature points and line segments.

In Section 4.3.2 we showed that in order to classify non-slippable regions in equivalence classes (and subsequently microtiles), one needs to compute all transformations that match parts of those regions r-symmetrically.

Lemma 5. Every non-slippable region is characterized by at least two non-parallel geometric edges in the triangle mesh. Note that those are actual edges in the geometry, not every triangle edge needs to be considered.

Proof. Consider a point $\mathbf{x}$ of the surface defined by the triangle mesh. Assume that there exists not more than one geometric edge intersecting the r-neighborhood of $\mathbf{x}$. Then it follows by definition that the r-neighborhood of $\mathbf{x}$ is either two-slippable or one-slippable. □

In the naïve algorithm for microtile detection (Section 4.3.2), corner features were used to compute candidate transformations. In a second step, symmetric regions were computed by transforming and matching the whole input mesh to itself. We will show that it suffices to compute a decomposition into partial symmetries only for the key points (corner features), which corresponds to computing microtiles w.r.t. r-symmetry for $r \to 0$. It is then possible to deduce the mesh segmentation into microtiles from this intermediate representation.

Lemma 6. It suffices to decompose the r-neighborhoods of all geometric corners and edges to obtain a surface decomposition into microtiles in the sense of Definition 4.

Proof. Consider a point $\mathbf{x}$ on the surface that is a non-slippable region. Non slippable regions consist of r-neighborhoods of corner features – we construct a corner feature at the closest point to each pair of non-parallel edges. Because we know which microtile the corner feature does belong to, we can determine at least one microtile, such that $\mathbf{x}$ is in the r-neighborhood of a point on the tile. This shows that we can reconstruct each non-slippable point of the input shape from the r-neighborhoods of its corner features. We therefore need to decompose the corner feature into microtiles to encode the complete surface. □

Limitations: Note that the microtile decomposition discussed here is not always equivalent to the microtiles w.r.t. to r-symmetry (Definition 6 and Theorem 1), because building blocks that do not contain a corner point are merged with the nearest microtile which does. Therefore some of the redundancies present in the input shape are not captured. On the other hand the new decomposition conforms to Definition 4 if we restrict the set of defining point-wise correspondences to the ones that map r-neighborhoods of corner features.

Note, that the slight redundancy in the new microtiles has no influence on the set of shapes that can be assembled from them, i.e. the possible shape variations remains the same and therefore the claim in Theorem 1 remains valid for the new decomposition. This follows from Lemma 5, which implies that all possible variations of rigid regions are represented by microtiles that contain at least two non-parallel edges with distance smaller than r.

Benefits: The above lemma allows for simplifying the r-symmetry detection in the following way. We no longer need to match the complete shape against its transformed copy in order to compute overlaps that define partially symmetric regions. Instead, we can identify and match all non-slippable (rigid) r-neighborhoods to each other. These are located within a distance r to at least two non-parallel line features, and can be centered at the point with the shortest distance to these two lines. This point is either an existing corner feature, or can be represented as a "virtual" one. The one-slippable edges can be processed afterwards, by matching their respective rigid neighbors at both ends.

Because we match only regions of radius r around points of interest, we eliminate the biggest performance bottleneck of the naïve algorithm, and replace it with a cheaper and easily parallelizeable alternative. In addition, assuming that we require a constant effort to match each pair of r-neighborhoods, the algorithm complexity becomes quadratic instead of cubic.

Restricting the decomposition to characteristic points has advantages not only in terms of the reduced running time of the algorithm. It also allows for exactly matching features to each other instead of having to compare voxelized parts of the surface, eliminating errors caused by miss-alignment of the discretized regions of the mesh. This is a critical advantage over the previous work, which suffered from inconsistencies at tile boundaries and required filtering to resolve errors.

Using the naïve method, the entire model was transformed via a given transformation matrix, trying to identify all matching segments of the shape globally. Here we generate the candidate transformations separately for each pair of interest points (corner features in our implementation) and compare small, local subsets of the geometry. This minimizes the influence of misalignment errors far away from the transformation center caused by the numerical representation of the entries of the matrix. We used single precision floating point numbers for both methods, and the second approach proved significantly more robust towards these precision issues.

4.4.2 Approximate Neighborhood Matching

An additional improvement of the r-symmetry detection stage of the algorithm can be obtained by matching the surface at pairs of key-points approximately. Instead of considering exact geometric matches, we compute how likely it is that the regions are r-symmetric. There are various ways to do that. We implemented a simple method that works on the voxelized geometry we use for the previous stages of the algorithm. Even though this voxelization approach was used to perform global matches in the naïve detection algorithm, here we are not interested in the boundaries of the matching regions, and do not have to consider the discretization artifacts, e.g. seen in Figure 4.4.

Let $N_r(\mathbf{x})$ be an r-neighborhood of a corner feature $\mathbf{x}$, and let $\mathbf{y}$ be an candidate for a r-symmetry under a transformation T. To perform a match, we need to compare the geometry inside $N_r(\mathbf{x})$ to $T(N_r(\mathbf{y}))$ (the geometry inside $N_r(\mathbf{y})$ transformed with T). We do this by matching all **fragments** in the voxelized region $N_r(\mathbf{x})$.

Definition 7. A **fragment** is a voxel-triangle overlap and is represented by a position (the projection of the cell center onto the triangle) and a normal (the geometric normal of the triangle). In other words, a fragment is a sample of an infinite plane aligned with a triangle intersecting a voxel. When comparing two corner points for geometric similarity, we consider all fragments in their r-neighborhoods.

Definition 8. Let f, g be fragments. We say that f and g are **matching fragments** if the angle between the normals of f and g is smaller than an **angle threshold** t_α (we used $20\,^\circ$ in our tests). Formally,

$$m(f,g) := \begin{cases} 1 & \angle(f.n, g.n) < t_\alpha \\ 0 & \text{otherwise,} \end{cases}$$

where $\cdot.n$ denotes the fragment normal, and $\angle(\cdot,\cdot)$ is the angle between two vectors.

Definition 9. We define the **distance** between a fragment f and a set of fragments G to be the distance to the closest matching fragment in G. Let $G_f := \{g \in G : m(f,g) = 1\}$ be the set of fragments in G that match f. Let $p(f,g)$ be the projection of the position $f.p$ of the fragment f onto the infinite plane

defined by $(g.p, g.n)$.

$$dist(f, G) := \begin{cases} \min\{|f.p - p(f, g)| : g \in G_f\} & \text{if } G_f \neq \emptyset \\ \infty & \text{otherwise,} \end{cases}$$

Definition 10. We define the **similarity** between two sets of fragments to be the percentage of matching fragments inside these sets, closer than a **distance threshold** t_d (we used $\frac{r}{10} + v_d$, where v_d is the length of the voxel diagonal). Let G denote a set of fragments, and f be a fragment. We define

$$M(f, G) := \begin{cases} 1 & dist(f, G) < t_d \\ 0 & \text{otherwise,} \end{cases}$$

and let F, G be two sets of fragments, then

$$Sim(F, G) := \max \left\{ \frac{\sum_{f \in F} M(f, G)}{|F|}, \frac{\sum_{g \in G} M(g, F)}{|G|} \right\}$$

In order to compute how closely two r-neighborhoods $N_r(\mathbf{x}), N_r(\mathbf{y})$ resemble each other, we compute $Sim(Fr(N_r(\mathbf{x})), Fr(N_r(\mathbf{y}))$, where $Fr(\cdot)$ gives the fragment contained inside the neighborhood. Because $Sim(\cdot, \cdot) \in [0, 1]$, we chose a threshold ($t = 0.9$ in our tests) and considered neighborhoods with similarity above it to be equivalent, which makes the respective center points r-symmetric.

The above formulation of similarity is similar to SIFT features [Low99], and allows for matching arbitrary sets of fragments, i.e. it is not restricted to individual r-neighborhoods. This allows for comparing larger subsets of the input surface to each other and distributing potential differences across larger surface area, allowing to identify equivalent microtiles even if the similarity of some pairs of their features is below the initial threshold t. We use this property later on to better match large building blocks resulting from the merging of multiple microtiles together.

A limitation of the similarity metric $Sim(\cdot, \cdot)$ involves transitivity. Given three sets of fragments A, B, C and a threshold t, having $Sim(A, B) < t$ as well as $Sim(A, C) < t$ does not necessarily imply $Sim(B, C) < t$. For microtile detection, we need the matches to form an equivalence relation. If the similarity metric is not transitive, the correspondence graph we compute might have missing edges. This can be caused by noise, deformations, or simply by numerical precision issues when matching geometry. However, since we assume an equivalence relation, and therefore know that each connected component of the correspondence graph must be a clique, we can add the missing edges. This is equivalent to ignoring, for some elements, that they are less similar than the allowed threshold t, if there are additional elements that are similar enough to each of the first two.

Note that the approximate matching of r-neighborhoods of key-poins can be used to match noisy or deformed input data. However, in order for this to

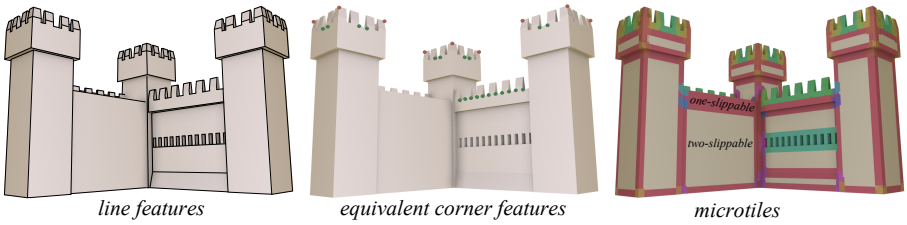


Figure 4.5: *Mircotile decomposition pipeline. First, all geometric edges are computed and later used to identify all corner features. Then r-neighborhoods of corner features are matched to compute equivalence cliques. Features and neighborhoods with the same outgoing equivalence transformations are merged into microtiles (third image). Tiles of the same shape have the same color. In the third image, two-slippable geometry is gray and one-slippable pieces are burgundy.*

work one must first find unique key-points inside each non-slippable region of the input model. While sometimes possible, but not easy in general (see [BBW+09b]). Since the main focus of our work is not feature detection, we limit our experiments to relatively clean input models, and use this matching method because of the added robustness to numerical errors and the ability to match geometry undergoing some small deformations, which we improve upon using the method we introduce in Section 5.5.

4.4.3 Algorithm Overview

Based on the above derivations, we can formulate a microtile extraction algorithm that operates on a finite subset of points on the input surface and therefore does not require global self-similarity matches. More importantly, the new discretization into corner features is invariant under rigid transformations, allowing to compute correspondences between pairs of individual elements rather than regions of unknown size. This was not possible to do with the naïve algorithm, because, in general, a rigidly transformed voxel will overlap more than one other voxel.

1. Perform slippage analysis of the model.

   We do this by computing a voxel representation of the input geometry, which we store in a uniform grid. We then compute all 1- or 2-slippable voxels.

2. Find key points and use them to compute candidate transformations.

   We compute all geometric edges and their intersections, which we call **corner points** or **corner features** (see Figure 4.5). We then compute all possible transformations that might map two corner features symmetrically. Similarly to the naïve algorithm we try to discard invalid transformations as early as possible.

3. Compute the pairwise correspondence of the corner features (see Figure 4.5).

We use the voxel grid computed in the first step of the algorithm to approximately match r-neighborhoods of corner points to each other. This gives us a distance, which we convert to a probability for a pair of corner features to be r-symmetric.

We propagate and make probabilities p consistent within cliques in the correspondence graph. We use a probability threshold (between 0.01 and 0.04 in our experiments) to determine equivalence and convert each connected component to a clique by "inserting the missing" edges (see Section 4.4.2).

An alternative approach for the same operation on noisy or deformed data can be implemented via spectral clustering similarly to Lipman et al. [LCDF10].

4. Compute microtile membership for the corner features.

   We only need to sift corner features into equivalence classes to obtain the decomposition. The remaining points of the surface are contained in the r-neighborhoods of the respective corners and edges.

   To sift corner features we employ the same approach as in the previous step of the algorithm: We compute the likelihood with which cliques of corner features share the same microtile class by comparing the relative locations of their elements (corner features).

5. Compute a segmentation of the shape into microtiles (see Figure 4.5).

   We already have a microtile decomposition for the corner features. The r-neighborhood of the latter together with the neighborhoods of their connecting edges cover the complete input surface, which allows us to segment it into building blocks.

Note that this segmentation ignores microtiles that do not contain a corner feature, but is nevertheless a valid decomposition in the sense of Definition 4.

The most significant difference with the naïve algorithm is the sparse discretization into corner features as opposed to voxels. A voxelized representation of the surface is still used to match potentially symmetric r-spheres around corner points, however these matches are not computed per voxel, but for a small number of corner features. This allows for one-to-one mapping between discrete elements, for which we compute self-correspondences. In the first algorithm, we had to compute per-voxel correspondences, which caused discretization artifacts on tile boundaries because each transformed voxel overlaps multiple other voxels. The new discretization allows us to eliminate the surface parts that are not assigned to a microtile (Figure 4.6).

4.4.4 Compact Transformation Representation

Real-life models often need to be decomposed into a set of microtiles such that many elements have global symmetries. For example a corner of a box or building is 3-way symmetric to itself. This produces redundancies in the set of transformations that map the feature to other surface points symmetrically. Namely,

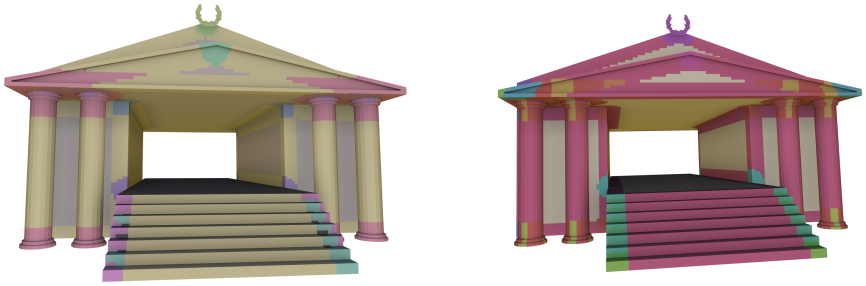


Figure 4.6: *Left: Output of the naïve decomposition algorithm for the courthouse model. The discretization artifacts on the stairs remain even after filtering. Right: The microtile decomposition is computed exactly per key point (geometric corners) and then transferred to the voxelized representation for illustration purpose The slight discrepancies in the two segmentations are caused by limitations in the naïve algorithm, which prevent accurate detection of correspondences between very small regions.*

the number of valid transformation between a pair of features is a multiple of the number of transformations that map each of the feature to itself. For the microtile decomposition we need to compute and store cliques of corner features. To reduce the computational effort and storage requirements we do not store all transformations that map pairs of points in a clique symmetrically. We can easily show that it suffices to to store one transformation for each feature point in the clique plus a single (and complete) set of transformations that maps each feature point to itself symmetrically.

More formally: In order to represent the equivalence class consisting of the r-neighborhoods centered at points $p_1, ..., p_n$, we need to find one symmetry transformation T_{ij} for each pair p_i, p_j with $i \neq j$. However if the r-neighborhood of p_i is symmetric to itself under multiple transformations (e.g. $120°$ rotations if p_i is the corner of a cube), then each of these latter transformations can be multiplied with T_{ij} and yield a symmetry mapping between p_i and p_j as well. We therefore avoid testing each of the redundant mappings, and discard some of the candidate matches early on.

In addition, we use this property to discard pairs of r-neighborhoods that cannot be symmetric, because they have different number of global self-symmetries. For example the corner of a box has 3 global symmetries, while the top of a square pyramid has 4, which means that one cannot rigidly map one onto the other symmetrically. In practice we were able to eliminate up to a third of the candidate transformations without having to perform additional tests. Because computing the geometric matches is the main computational bottleneck and the number of potential matches is given by the number of candidate transformations, we reduced the overall run time of the decomposition with 30% for test models like the courthouse in Figure 4.6 and the castle in Figure 4.5.

Another optimization opportunity presents itself when storing the computed equivalence classes of r-neighborhoods and their symmetry transformations. Instead of storing one transformation per pair equivalent elements, we can keep

only linear number of matrices. The transformations should map p_1 to $p_2, ..., p_n$. All other mappings can be reconstructed as a composition of the stored ones if necessary. In the correspondence graph analogy, this means that instead of storing all edges inside each node clique, we can remove maximal amount of edges, such that the cliques remain a connected component.

A useful approach to eliminate further candidate symmetry transformations of individual features is to classify the outgoing line features by length and to ensure that each transformation maps edges shorter than the symmetry radius to edges of the same length. Edges longer than the radius should be mapped to counterparts not necessarily equal in length, but not shorter than the radius of symmetry.

With the above optimizations we were able to significantly reduce the number of geometric matches we have to compute. For the courthouse model (Figure 4.6) we could eliminate 2.92 million out of the initial 3 million possible matches before having to perform an actual distance test on the voxelized r-neighborhoods around pairs of corner features. For the model in Figure 4.5 we could eliminate 12.39 million transformation candidates and only had to perform around 160 000 matches.

4.4.5 Parallel Implementation

An additional advantage of matching individual regions around corner features is related to the implementation of the algorithm. As opposed to the naïve approach where symmetric regions were computed by matching the entire shape to a transformed copy, applying the insights from Lemma 5 and Lemma 6 allows us to use local matches instead. We compare the geometry inside (multiple) pairs of spherical r-neighborhoods, and these operations are easier to distribute across multiple (CPU and/or GPU) threads. In addition a complex hierarchical structure like an octree is no longer necessary – it is more efficient to use an uniform grid an match all voxels inside the two r-neighborhoods of the pair of corner features.

4.4.6 Results

We implemented and tested the efficient microtile detection algorithm on a PC with an Intel Core i7-3770K CPU with 4 cores running at 3.5GHz and an NVIDIA GeForce 660Ti GPU. We used the GPU to compute a scene voxelization and to perform all geometric matches in the third step of the algorithm (see Section 4.4.3). The latter proved to be slightly faster (20%) compared to a parallel CPU implementation.

The efficient r-symmetry detection algorithm reduced the overall running time significantly – up to two orders of magnitude compared to the naïve version. This allows for efficiently decomposing even non-trivial input models that have partial correspondences in the orders of millions. The resulting decomposition does not suffer from artifacts at tile boundaries and the better efficiency allows

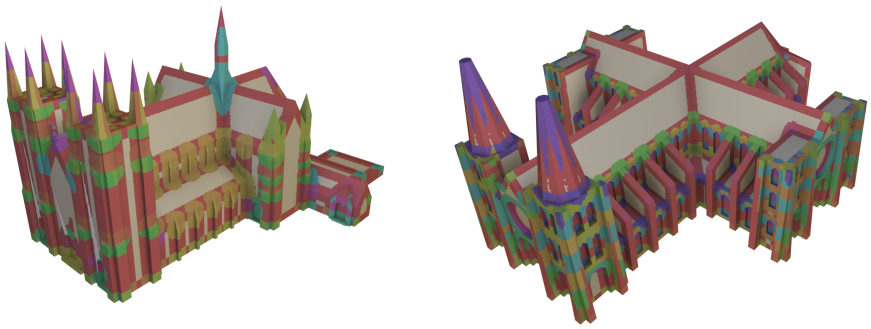


Figure 4.7: *Microtile decompositions of two test models downloaded from thingiverse [Thi]. The cathedral on the left was decomposed in 38s into 931 microtiles of 527 classes. The church on the right was decomposed in 3min into 2276 microtiles of 395 classes.*

for computing arbitrarily small microtiles. Note, that the visualization artifacts in the figures (e.g. Figure 4.7 and Figure 4.8) are caused by the use of a low resolution global 3D texture to color the individual microtiles. Computing the exact boundaries of the building blocks is a difficult problem, which we discuss and solve later in Chapter 7.

We evaluate the new decomposition algorithm on mostly clean input models. The approximate neighborhood matching around individual features allows for handling slight deformations and typical inconsistencies in the geometric representation caused by the floating point representation of vertices and transformation matrices. In Chapter 5, we extend the decomposition approach further and also handle very small deformations that can be caused by noise or artist errors.

4.4.7 Discussion

In this chapter we showed how to lift important limitation of the microtile analysis introduced in [KBW+12]. The main theoretical insight here is that although r-symmetry and r-similarity are defined per point, it suffices to analyze a finite amount of surface elements in order to perform a complete analysis (and segmentation) of the input surface. The elements we consider here are geometric edges and corners and slippable regions of the input surface.

With the algorithms for r-symmetry detection and microtile decomposition discussed here, we introduce a novel and practical tool for shape analysis. The insights we employed in order to develop the efficient extraction algorithm allow us to develop a practical tool for partial symmetry detection and use it as a starting point of several interesting application for mesh compression (Chapter 6) and procedural modeling (Chapter 7). While there certainly are other methods for symmetry detection and different definitions of building blocks, so

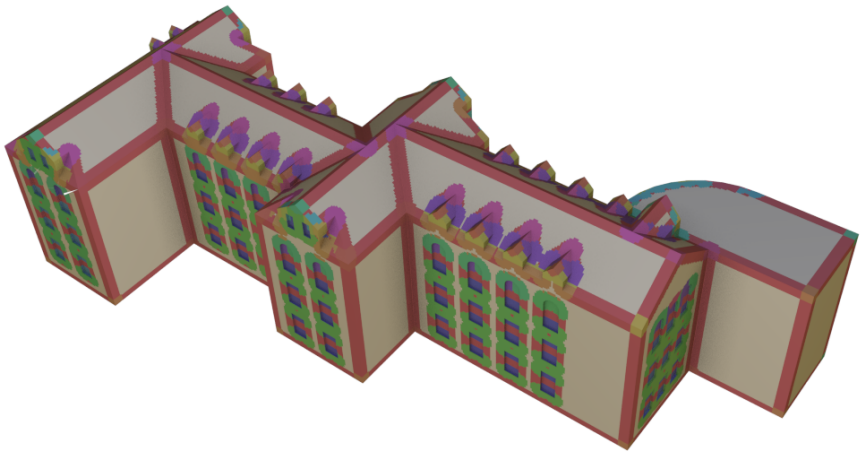


Figure 4.8: *It took 28s to decompose the above building on into 921 microtiles of 196 classes.*

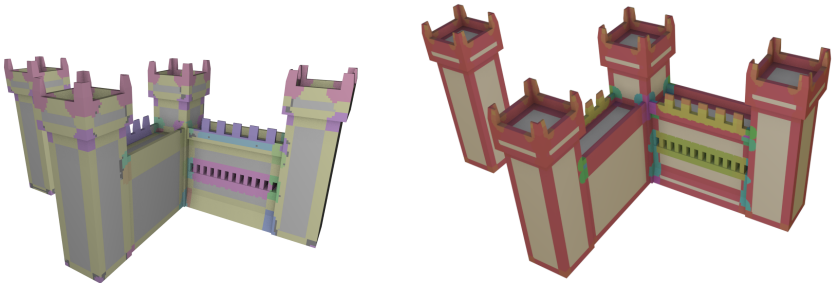


Figure 4.9: *Left: Output of the naïve decomposition algorithm for a castle with 4 towers. Run-time was approximately 1 hour. Right: the new algorithm is able to compute more detailed decomposition in 52s.*

far only the microtiles, introduced here, allow for systematically computing and characterizing a concrete family of shape variations – an essential property for applications in the area of shape analysis and inverse procedural modeling.

We improved the reliability of the naïve decomposition approach, which suffered from fundamental problems related to the global discretization and self-matching routines. This allowed us to reliably compute detailed decomposition into microtiles w.r.t. r-symmetry, thus making it possible to perform complex shape analysis with the resulting building blocks. Furthermore, we drastically improved the efficiency of the detection algorithm, which allows its use as a pre-processing stage for modeling applications.

In the following chapter we can build on the results presented here and develop a mechanism for building block simplification. With them we address the complexity of the decomposition and use it to compute optimal building blocks for various applications.

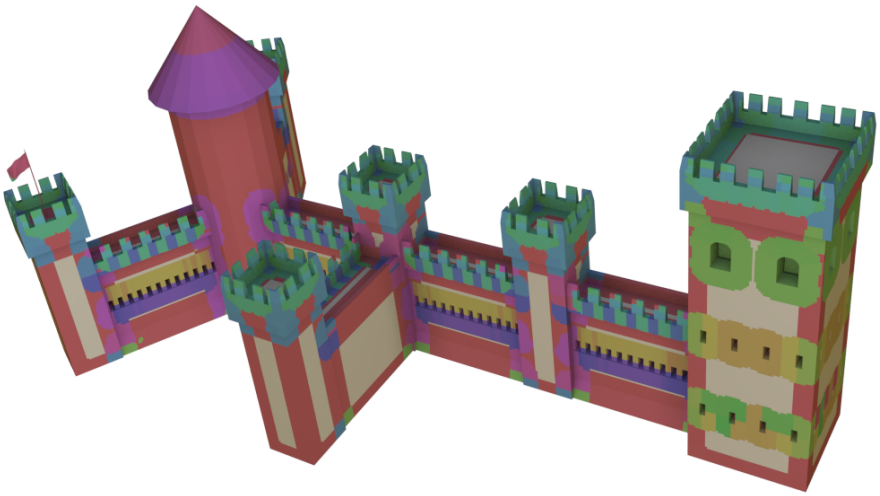


Figure 4.10: *This castle model was decomposed in 4min into 2078 microtiles of 962 classes. We tested 2 million r-symmetry candidates – triplets of two features and a transformation.*

5 | Tiling Grammar Simplification

So far in this thesis, we have described a system for microtile analysis of shapes. In Chapter 3 we introduced the theoretical foundations, defined the building blocks we call microtiles, and proved that they can be used to describe infinite families of r-similar shapes. In Chapter 4 we discussed how to implement a robust, high-performance microtile analysis system for triangle meshes. To this end, we address various practical limitations that previously prevented robust microtile extraction.

We now want to use the symmetry information encoded by the microtiles for applications like inverse modeling of shapes, shape analysis, and mesh compression. The microtiles can be used in each of these scenarios without modification, however they are not necessarily optimal for any of these purposes. The building blocks minimize the number of r-neighborhoods of points needed to represent an input shape and all shapes r-similar to it. This is the case, because our definition of redundancy (or similarity) is based on r-symmetry of surface points. While theoretically sound, in practice, this approach can lead to oversegmentation of the input shape, making the building blocks too expensive to store (for mesh compression), or generating too many pieces to allow intuitive assembly into shape variations (inverse modeling).

In this chapter, we introduce a method for tiling grammar simplification. Our goal is to convert the initial microtile decomposition into a new set of building blocks, better suited for a pre-defined application. In other words, we modify the microtiles aiming to obtain an optimal set of building blocks according to some cost function.

5.1 Related Work

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. So far in this thesis, this concept was generalized by "microtiling" the surface through the set of all possible symmetric curve pairs. Unfortunately, this yields a grammar containing infinitesimally small, slippable pieces in addition to a large number of small rigid building blocks. Because of that, applications like mesh compression and manufacturing remain challenging.

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

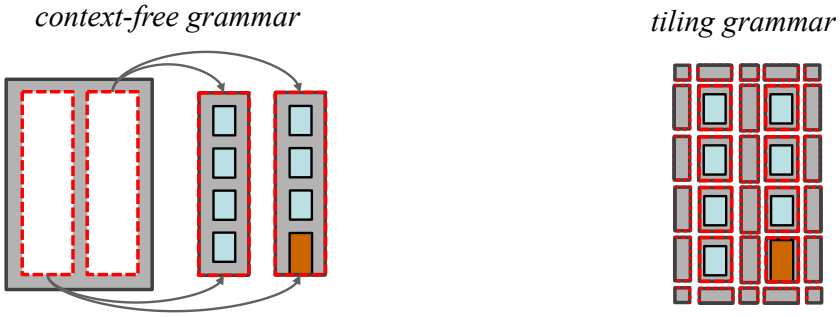


Figure 5.1: *Example shape grammars in 2D. The context-free grammar consists of hierarchical sets of rules, making it easier to create shapes. On the other hand, tiling grammars have greater expressive power and, in the case of the microtiles discussed here, have building blocks that correspond directly to the partial symmetry structure of the resulting shapes.*

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. 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 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.

Obviously, this problem tightly related to recent techniques for inverse procedural modeling [ŠBM+10, BWS10, KBW+12, JTRS12, ZCOM13, MWZ+13, LVW+15]. 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. On an abstract level, these approaches can be understood as automated constructions of **tiling grammars** [KBW+12, LVW+15] that build graphs of nodes (representing the building blocks) that are connected through edges conforming to local connectivity constraints (constituting the rules of the tiling grammar). The individual approaches differ in how exactly the tiles and rules are obtained (for example, context free approaches [BWS10] are a special case of general tiling grammars [LVW+15]), and what procedure is employed to instantiate and geometrically embed the graphs.

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 they only consider context-free shape grammars, i.e., all assembly rules must be hierarchically structured (see Figure 5.1). 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

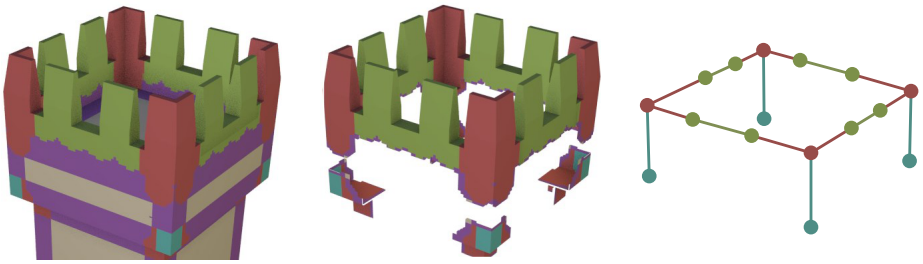


Figure 5.2: *Mircotile decomposition (left), non-slippable microtiles (middle) and the resulting graph (right). Equivalent building blocks, graph nodes, and graph edges are colored identically.*

and simplifying this type of information is a valuable tool of its own, given the importance of partial symmetry in modern geometry processing algorithms.

5.2 Microtile Graphs

The initial microtile decomposition we obtained is typically too detailed to be practical for a wide range of applications. For example, the model in Figure 4.8 is decomposed into 921 tiles of 196 different types (excluding continuous parts). If we consider mass-production via molding as a possible application of our analysis, the current representation becomes quite inefficient due to the large number of molds necessary to produce each type of building block. The main result in this chapter is a graph-based algorithm for optimizing the microtiles defined in the previous section. Our framework will allow for efficiently transforming the structure induced by the partial symmetries into a decomposition that is better suited for a particular application (e.g. mass-production of the building blocks). We obtain the improved building blocks by sacrificing some of the expressive power of the microtiles, i.e., we trade off the number of possible shape variations that can be constructed out of the new building blocks for a simpler decomposition of the input model.

Definition 11. Microtile Graph: We define a microtile graph (Figure 5.2) to be a representation of the microtile decomposition. The graph nodes correspond to rigid (non-slippable) microtiles, and two nodes are connected if they have an overlapping spherical r-neighborhood, or if they are connected to the same 1-slippable edge.

This means that a graph edge can represent ether a 1-slippable edge of the input mesh, or intersecting r-neighborhoods of two rigid building blocks. We do not include 2-slippable regions in the microtile graph, but, if necessary, we keep track of the amount and type of their surrounding microtiles.

Observe that the correspondence of the graph nodes with microtiles allows for transferring equivalence relations from geometry to graph nodes and subsequently to graph edges (see Figure 5.2).

Definition 12. Equivalent Nodes: We call two nodes equivalent nodes whenever their corresponding microtiles are of the same class (type), i.e., the nodes are equivalent if and only if the respective microtiles are equivalent.

Definition 13. Equivalent Edges: We consider two edges $e_1 := (a_1, b_1), e_2 := (a_2, b_2)$ to be equivalent if a_1, a_2 and b_1, b_2 are from the same equivalence classes respectively and the relative transformation from a_1 to b_1 is the same as the one that transforms the local coordinate frame of a_2[1] into the coordinate frame of b_2. In other words, equivalent edges describe equivalent geometric relations between equivalent pieces.

Implicit grammar encoding: Importantly, the microtile graph itself implicitly encodes the tiling grammar that describes a whole family of shapes. 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.

The abstract graph we propose is based on a given symmetry-based decomposition of an input surface. We already specified a decomposition – microtiles for point-wise r-symmetry, however the general approach presented here is not limited to this particular decomposition. The latter can be substituted by any other symmetry-induced segmentation into recurring or exchangeable pieces (e.g. the user-defined segmentation used in [LVW+15] and Figure 5.6).

5.3 Microtile Cost Models

There are applications, for which the microiles and their structure are not optimal. After all, the decomposition only encodes redundancies due to partial symmetries. We are interested in modifying the microtile graph in order to optimize the building blocks for several different applications. One possible modification is collapsing sets of equivalent edges. Another possibility is to simultaneously deform all instances of a class of tiles such that it matches (resembles closely) the instances of a different class.

Because we want these operations to lead to some improvement we introduce the notion of a cost model associated with the graph and the operations we perform on it. Different applications can pose completely different (even contradicting) requirements for the nodes and structure of the graph.

Hence, we derive specific cost function and optimization strategy separately for each application. However, the approach, in general, is to define a global **graph cost** and perform graph transformations that minimize it. The global cost

[1]The local coordinate frame of each microtile is computed from its geometric corner features. We take into account their coordinate frames from the initial pairwise matching step (step 2) of the microtile detection algorithm in Section 4.4.3.

defines how well the individual pieces can perform a certain task (e.g. compress the geometry of the input mesh).

In Chapter 6 we discuss a cost model that reflects the memory required to store the geometric representation of the microtiles. Therefore, a decomposition with a minimal cost can be used for symmetry based, loss-less mesh compression.

Chapter 7 discusses a more challenging problem: We derive a cost model for 3D manufacturing. We express various properties important for the physical realization of assemblable 3D constructor pieces that can be used for inverse modeling of shape families. As a result we convert the microtile decomposition into a segmentation of a few, and therefore cheap to mass-produce, pieces that can assemble a number of shape variants - just like the popular construction toys - LEGO.

In this chapter we focus on the general approach of modifying and optimizing microtile grammars and discuss the choice of appropriate cost function later. Therefore, we first consider a simple example of a cost function that allows to trade-off **redundancy** captured in the decomposition for the **simplicity** of the representation.

Simplicity: One way to derive a simple explanation for the structure of a given shape is to find a decomposition into minimal number of different recurring pieces. In other words we would like to minimize the number of microtile classes. Let $t \in Tiles$ be a non-slippable microtile. Let $N_{eq}(t)$ be the number of microtiles equivalent to t (tiles of the same microtile classes). Let $N_{Classes}$ be the total number of different types of microtiles:

$$N_{Classes} := \sum_{t \in Tiles} \frac{1}{N_{eq}(t)} \tag{5.1}$$

Let $N_{InitialClasses}$ be the number of classes in a canonical microtile decomposition. We define the cost for simplicity for a given modified decomposition C_S as:

$$C_S := \frac{N_{Classes}}{N_{InitialClasses}}, C_S \in (0, 1] \tag{5.2}$$

Redundancy: The redundancies represented in the microtiles are measured by summing up the areas of one representative tile for each microtile class.

Let $SA(t)$ denote the surface area of a tile. Let $N_{sym}(t)$ be the number of different transformations that lead to global self-similarities of tile t. We define the **redundancy cost** C_R:

$$C_R := \frac{\sum_{t \in Tiles} \frac{SA(t)}{N_{eq}(t) N_{sym}(t)}}{\sum_{t \in Tiles} \frac{SA(t)}{N_{sym}(t)}}, C_R \in (0, 1] \tag{5.3}$$

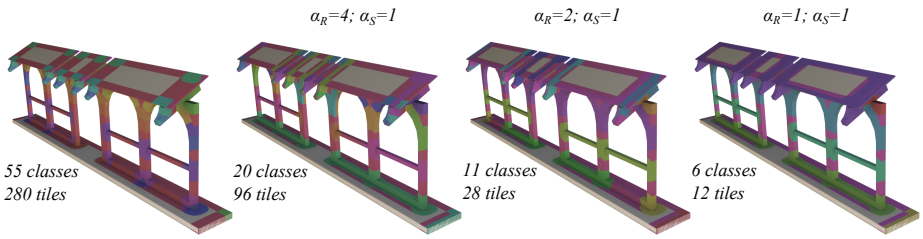


Figure 5.3: *A microtile decomposition of a fence model (left) and optimization examples using the basic cost funtion $Cost := \alpha_R C_R + \alpha_S C_S$ to control the amount of simplification. From left to right: C_R rises from 0.1 to 0.4; C_S drops from 1 to 0.14.*

In other words the redundancy term C_R is equal to ratio of the combined surface area of points with unique r-neighborhoods divided by the total surface area of points with rigid r-neighborhoods.

Note that we only need to consider non-slippable microtiles, because the slippable pieces have zero area. Also note that while $C_R = 0$ means that the points on the model with rigid r-neighborhoods have zero measure, which is not the case in general. Usually, because the symmetry radius r is larger than 0, the rigid parts of the model have area larger than zero, which makes C_R positive.

Lemma 7. The initial microtile decomposition has minimal redundancy cost C_R.

Proof. Assume the opposite – there is a non-zero area surface patch that is contained in instances of microtiles from different classes. This implies that there is a (partial) match between the two building blocks which contradicts Lemma 1. □

In other words, any combination of microtile instances of different microtile classes is a minimal subset of the input shape that contains the selected r-neighborhoods (modulo global self-symmetries of pieces). The above considerations imply that the smaller the cost for redundancy of a given segmentation, the closer it will be to the canonical microtile decomposition we compute in Chapter 4.

Finally, we can define a generic cost function that measures the quality of a given decomposition w.r.t. to simplicity and redundancy in the sense defined above:

$$Cost := \alpha_R C_R + \alpha_S C_S, \tag{5.4}$$

where α_R and α_S are used-defined weights. To optimize the cost of a microtile segmentation we need to be able to modify it. In the following, we discuss two possible approaches for that.

5.4 Edge Collapsing

In order to improve the microtile decomposition, we modify the structure of the segmentation of the shape into building blocks. An efficient way to perform this operation is to collapse edges of the microtile graph. Each collapse will merge two building blocks with each other. In order to preserve the symmetry structure in the new representation, we only collapse whole equivalence classes of edges. 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 in the graph. This is very important, because it prevents the introduction of unnecessary redundancies in the modified decomposition.

Note that collapsing a set of graph edges does not change the input geometry, only its segmentation into microtiles, and is therefore a loss-less operation with regards to the geometric representation of the input object. Also note that although we try to improve the actual building blocks, we perform transformations directly on the microtile graph. This makes the transformation easy to perform and very fast.

5.5 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.

On the other hand, our graph simplification approach facilitates discovering redundant regions in a hierarchical fashion. We can take advantage of this property by matching pairs of tile classes after merging them. Increasing the size of the segments reduces the importance of local errors or deformations. Our similarity metric allows for identifying geometric regions that are 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 for distributing the local errors over a larger surface area and discover additional redundancies in the simplified decomposition.

In a scenario where preserving the input geometry is not necessary, the symmetry structure encoded in the microtile graph can be simplified by modifying the geometry of the building blocks. One can combine two or more classes of microtiles into a single one. One way of doing this is to discard all instances of class B, and replace them by instances of class . Alternatively one can compute a replacement class C, by deforming the instances of A and B.

Including tile replacement operations during the optimization made our algorithm robust to local deformations caused by artist errors (Figure 5.4) and discretization artifacts in curved shapes in some models (see Figure 5.5). Oval parts in other models (e.g. the columns in the courthouse model in Figure 4.6),

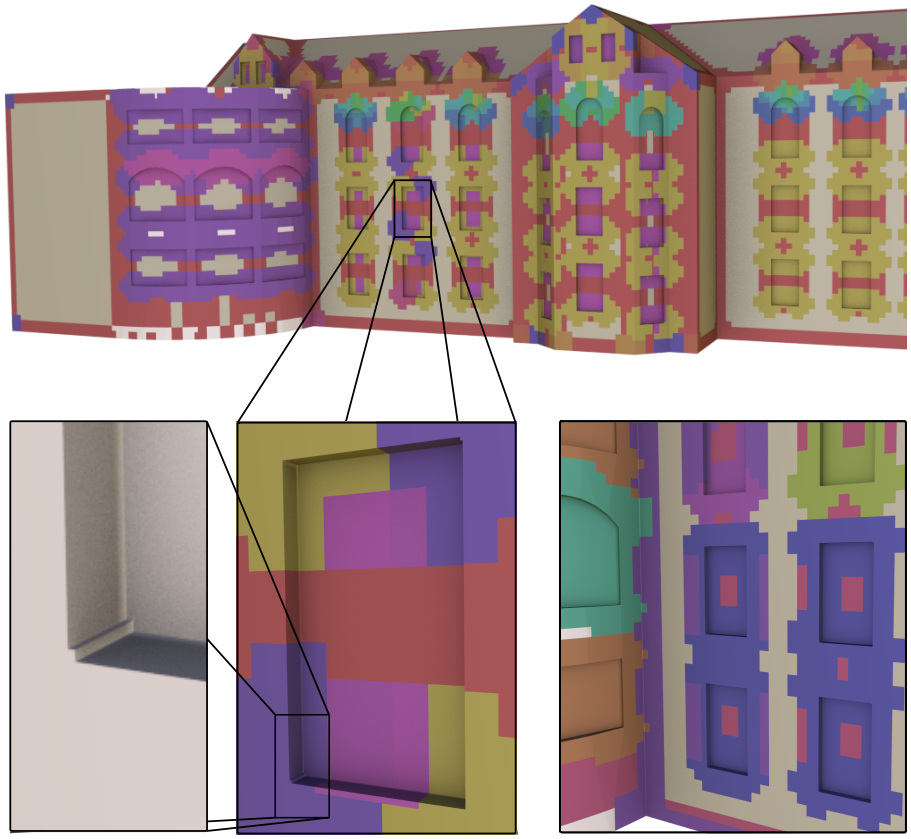


Figure 5.4: *The corners of several windows on this building differ from the rest, probably due to a modeling error. Lower right: Merging the microtiles into larger building blocks allows for approximately matching these regions despite their discrepancies.*

could be identified without the additional use of tile replacement.

Even if we are allowed to modify the input, replacing arbitrary pieces of geometry seldom results in a meaningful shape. To enforce only meaningful modifications, we only replace a tile if the operation preserves its boundary with the rest of the shape. This guarantees that performing tile replacement will not introduce holes or discontinuities in the shape, if there is no evidence that such anomalies are acceptable.

Note that a more general operation is to replace entire sets of microtiles at once. Finding appropriate sets amounts to finding sub-graphs of microtiles with matching boundaries. This is a generalization of the approach by Bokeloh et al. [BBW$^+$09b], in which the authors find such boundaries randomly, without computing microtiles. However, each such operation is equivalently performed by collapsing all edges in the respective sub-graphs and replacing the resulting bigger microtiles afterwards.

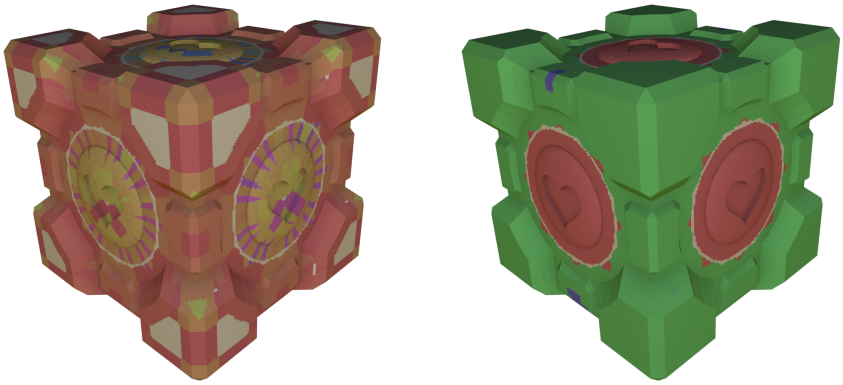


Figure 5.5: *Replacing tile classes during simplification facilitates fixing errors in the initial decomposition. In this example the parts of the model with complex curvature can be identified as similar, even though some of their r-neighborhoods do not match precisely.*

5.6 Optimization Approach

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 introduce changes in the node connectivity. This makes the behavior of the cost function for graph edges difficult to predict – collapsing an edge can increase or decrease the overall cost by a different amount, depending on which other edges have been previously collapsed. Therefore it is not possible to express the minimization objective as an easy to minimize, convex (or even continuous) function.

On the other hand, given a set of (non-equivalent w.r.t. Definition 13) edges that will be collapsed, the resulting graph is the same regardless of the order in which the operations are performed. This shrinks the search space for an optimal solution, and instead of having to find an optimal sequence of transformations, one needs to determine an optimal subset of the graph edges to collapse. The latter considerations also imply that an optimal solution is computable, due to the finite number of possible transformation for a given graph.

Because the graph transformations do not involve any complex geometric computations, it is possible to perform exhaustive search for quite large input sizes in reasonable time. Nevertheless, we tried to find more efficient search algorithms. The best results we obtained were by sampling the space of possible solutions. We performed a series of random edge collapses. Each series was terminated randomly via Russian Roulette with probability $\frac{1}{N_{Classes}}$ ($N_{Classes}$ is the number of different microtile classes (types)). For each model we performed roughly about $N_{Classes}$ searches and kept the graph with the best cost. Although this approach will only find an optimal solution with some probability, in our tests we could always improve the initial segmentation significantly.

We 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 5.3) smallest while at the same time reducing the number of building blocks and potentially the classes of microtiles (C_S, Equation 5.2). This method was faster compared to the random search, which delivered slightly (around 15%) superior results. Further discussion on optimization strategies is outside the scope of this chapter, because the results depend on the choice of the cost function. Here, we demonstrate that edge-collapse algorithms can be used as a general method for optimization of shape decompositions.

Sometimes we were able to further improve the solution by re-starting the sampling process and initializing it with the best solution from the previous run, but the improvements were never significant. Also note, that although this stage of the algorithm works without user interaction, this does not have to be the case. One can also compute a number of solutions with cost close to the optimal and let the user choose one.

Another advantage in terms of complexity is gained by the fact that we perform collapses of sets of equivalent edges (i.e. edges that connect tiles of the same types). In other words whenever we consider collapsing an edge, we collapse a whole set simultaneously. This makes the number of potential solutions exponential in the number of pairwise non-equivalent edges instead of the total number of edges. In Figure 5.2 we have 16 edges but only 3 possible collapse operations because of equivalence. If a set of equivalent edges to be collapsed form a cycle, we always collapse the maximum number of edges, which results in a simpler resulting graph.

5.7 Shape Variations

One of the important properties of microtiles with respect to r-symmetry is their ability to describe families of (r-similar) objects. However, modifying the grammar can reduce the amount of related shapes that are assemblable from the building blocks. It should be noted that processing any finite set of shapes simultaneously results in building blocks that construct each of the input shapes. In the following two sections we discuss the more challenging problem of generating (infinitely many) variations of a single input object.

It is possible to restrict the edge collapses in order to guarantee that the resulting pieces keep their property of characterizing the space of r-similar objects. This ensures that the set of shapes that can be constructed by the building blocks does not shrink after the transformation. However, we preferred to trade-off some of the expressive power of the building blocks to improve other properties like size, number of occurrences, and simpler overall structure of the shape, expressed by the cost function (see Equation 5.4). Therefore, in addition to collapsing safe edges we allow three operations that can potentially reduce the number of shapes constructable by the microtiles:

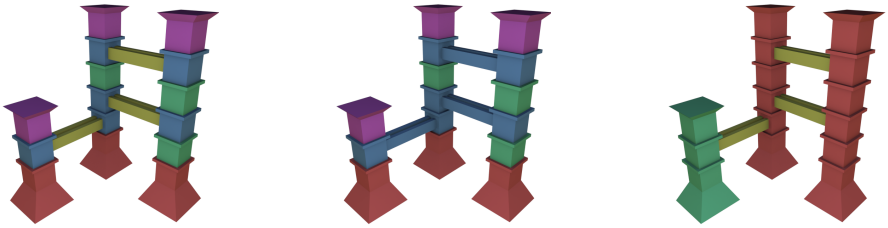


Figure 5.6: *Left: A model is decomposed manually in 18 pieces of 5 types. Middle: The optimized version consists of 12 pieces of 4 types. Right: Allowing replacement of approximately matching tiles further reduces the complexity to 6 pieces of 3 types.*

- We compact microtile classes by merging pairs of microtile instances from the same class. This can affect the variations in the size of grid-like repetitive patterns of tiles.
- 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 where the corresponding edges are longer or shorter.

Note that the majority of transformations we perform do not fall in any of the above categories and therefore preserve the expressive power of the decomposition.

If the desired output of the algorithm is a set of microtiles that can construct a given finite collection of r-similar objects, one can perform the microtile decomposition and the subsequent optimization simultaneously on the whole collection. This will guarantee that the resulting tiles will construct each exemplar in the target collection regardless of the simplification process. However an even more interesting challenge is to compute manufacturable microtiles that can construct an infinite set of shape variations (see Chapter 7).

5.8 Evaluation

In this chapter we evaluate the simplification algorithm briefly with the simple cost function we introduced. We provide more detailed analysis in the next two chapters, where we derive application-specific cost models and use them to decompose shapes into pieces that can be used for mesh compression or 3D manufacturing.

The main result we can provide already with the simple cost model (Equation 5.4) is the ability to simplify a microtile decomposition, by sacrificing some of the redundancy in order to reduce the number of pieces required to represent the shape.

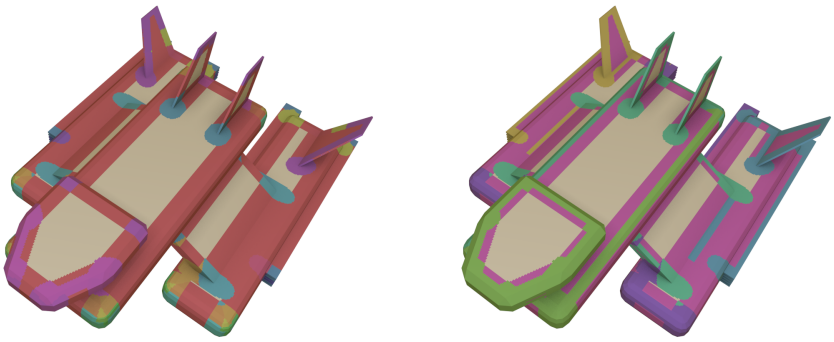


Figure 5.7: *Original (left) and optimized (right) decomposition of a spaceship model. The number of tiles was reduced to 7 from 319 and the types of tiles was reduced to 6 from 111.*

In Figure 5.3 we demonstrate how re-weighting the cost function terms enables intuitive control of the amount of simplification – the bigger the weight α_R for the redundancy term, the more pieces and types of pieces we obtain.

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 5.6. We show the optimal results without and with tile replacement operations (Section 5.5). Approximate replacement allows for identifying the entire red columns as "identical" if the discrepancies between the two are not significant. For the example in the figure, we allowed 10% of the surface area of one of the pieces to mismatch the other.

In general, using the graph simplification approach presented here, we are able to simplify the initial microtile decomposition substantially, while preserving significant amount of the symmetry information encoded in the initial decomposition. This indicates that graph-based tiling grammar simplification algorithms can be practical methods for improving symmetry-induced shape decompositions. We will further validate this claim in the following two chapters, by deriving appropriate cost models for the simplification method presented here and adapt the optimization process for specific applications.

5.9 Conclusion

The method introduced in this chapter is an important corner stone for detection and analysis of rigid symmetries. Unlike most of the related work in the area, which deals mostly with context-free grammars for shape generation, we address the problem of optimizing the more general context sensitive tiling grammars that naturally describe partial symmetries. On its own, the simplification algorithm allows for generating symmetry-aware shape decompositions

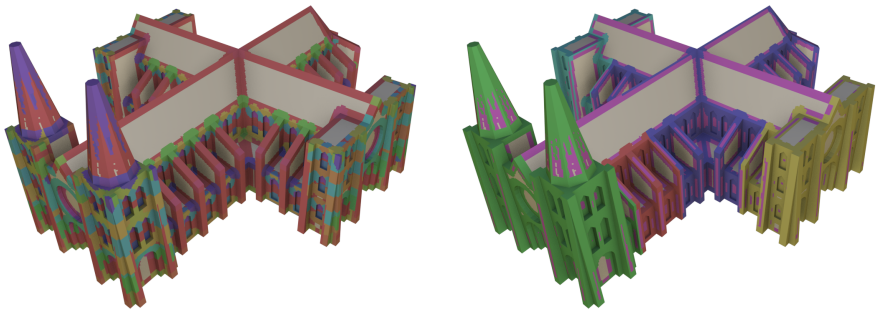


Figure 5.8: *Microtile decomposition of a church model. Left: the initial decomposition into 2276 microtiles of 395 classes. Right: after optimization, we got 8 microtiles of 5 classes. We used the cost function from Chapter 7 for this test.*

with controllable complexity. However, this approach will become more interesting when combined with specific cost functions, allowing to compute optimal shape decompositions for applications like mesh compression or 3D manufacturing.

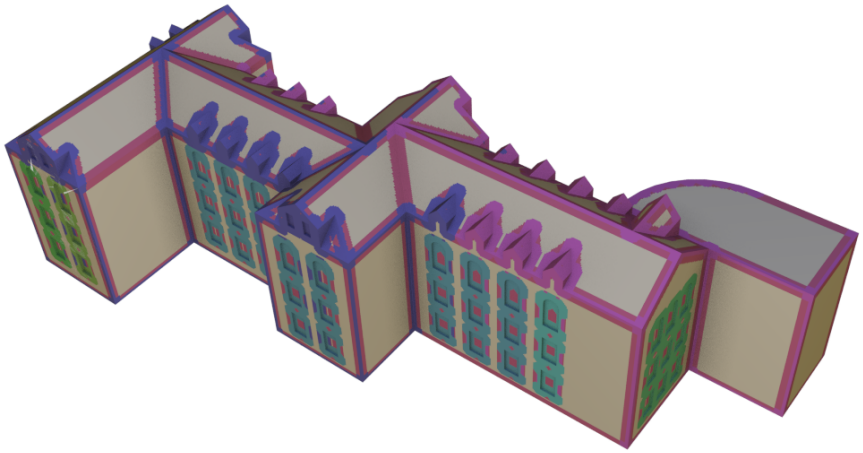


Figure 5.9: *After optimizing the microtiles for the building above, we reduced the number of tiles from 921 to 30 and the number of classes from 196 to 17. We used* $\alpha_R = 1$ *and* $\alpha_S = 1$ *for this test.*

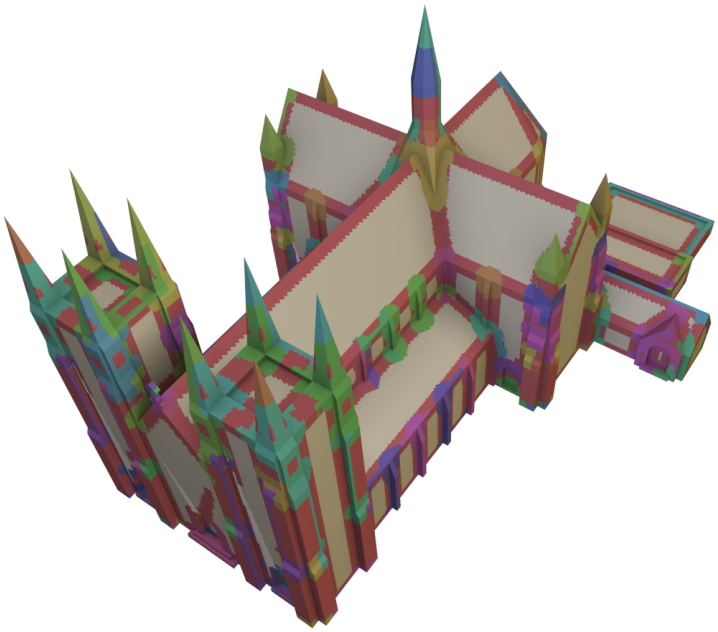


Figure 5.10: *After optimizing the microtiles for the cathedral model, we reduced the number of tiles from 931 to 206 and the number of classes from 527 to 135. We used* $\alpha_R = 3$ *and* $\alpha_S = 1$ *for this test.*

6 | Mesh Compression

In this chapter, we discuss the use of microtile decompositions for loss-less mesh compression in the context of real-time ray tracing. We use partial symmetries to inversely compute a 2-level hierarchical representation of the scene geometry and apply the optimization technique from Chapter 5 to obtain a scene representation with minimal memory footprint.

6.1 Ray Tracing

We are interested in using ray tracing in the context of physically-based rendering algorithms. This means that we want to generate 2D images of a virtual 3D scene. The rendering process is analogous to taking a digital photograph, and ray tracing is used to compute light paths for energy originating in the light sources that is transported to the virtual camera.

Formally, the operation of tracing a ray is defined as a query over a geometric scene. Given a ray with an origin o and direction $\vec{d}$, and a scene consisting of geometric primitives (e.g. triangles), we determine whether there is a primitive in the scene that the ray intersects, and optionally which of them is the primitive closest to the origin. With other words, we want to know if there is a non-negative scalar value t, such that $o + t\vec{d}$ is a point in space that belongs to a given primitive P in the scene.

$$\left\{ t \mid \exists P \in scene\ s.t.\ (o + t\vec{d}) \in P \right\} \neq \emptyset \tag{6.1}$$

We may optionally look for the first primitive intersected by the ray:

$$\min \left\{ t \mid \exists P \in scene\ s.t.\ (o + t\vec{d}) \in P \right\}. \tag{6.2}$$

To be able to answer such queries, one only must be able to perform an intersection test between a ray and a geometric primitive. Given a ray one can test every input primitive for intersection to determine if there is one. Even though this can be done in linear time it is very inefficient even for small scenes. This is why high performance ray tracing implementations rely on **acceleration structures**. These are search structures (the spatial analogue of dictionaries) that subdivide space into cells or nodes. During ray traversal, only primitives contained in nodes (or cells) of the acceleration structure that are intersected by the ray are tested. In this way some cheap traversal calculations help eliminate large amount of intersection candidates – those contained in cells that do not intersect the ray.

The way ray tracing is related to performing a search, acceleration structures for ray tracing are related to general search structures like dictionaries. For ray tracing one uses spatial structures that subdivide the space and store the geometric primitives in their cells. A Binary Space Partition tree (BSP tree) for example is the analog of a binary search tree. It partitions space hierarchically via planes and stores the primitives in its leaves. Instead of testing each ray against each primitive for intersection, one first traverses the tree to find all leaves intersected by the ray and tests only the primitives stored in those leaves. In this way some cheap traversal calculations help eliminate large amount of intersection candidates – those that are not contained in leaves intersected by the ray.

Because tracing a ray is a global operation – one needs to have access to all geometric primitives, the entire scene must be stored in physical memory during rendering. Therefore, the amount of storage space usually limits the size and geometric complexity of the input scenes. Note that in most rendering scenarios, the geometric models are only a part of the scene description that needs to be stored in memory. Assets like textures and material descriptions constitute a significant portion of the memory footprint of a typical scene. Nonetheless, reducing the size of the geometric representation of the surfaces is beneficial and can be used to work around memory bottlenecks.

6.2 Geometry Instancing

A well understood technique to reduce the memory footprint of massive geometric models is called **instancing**. The scene can be generated out of multiple copies of the same (or similar) objects, each of them is stored once in memory and a virtual copy is placed for each **instance** – each occurrence of the original in the scene [DMS06]. The advantage is that representing the instances only requires to place a bounding box surrounding the replicated geometry and record a transformation matrix and a pointer to the original. A two-level acceleration structure is computed – the top level is build over the bounding boxes of the instances, and for each original exemplar there is a separate acceleration structure (see Figure 6.1). During rendering, rays that are likely to intersect an instance are transformed with the inverse of the transformation matrix and then traverse the second-level spatial subdivision containing the original. Therefore, the geometry for the copies never has to be generated or stored.

Usually, the geometric objects that are replicated via instancing are static. This alleviates the need for expensive refitting of the acceleration structure containing the original object. However the copies can be transformed with rigid transformations, allowing the scene to be animated with rigid motion [GFW+06]. Dynamic scenes like these are typically supported in real-time ray tracing engines by rebuilding the top-level of the spatial structure which is invalidated after an object changes its position in the scene. This can be performed fast enough if the number of objects stored in the first level of the acceleration structure is not too large. Rebuilding the second level is not necessary in general, and can be overly expensive.

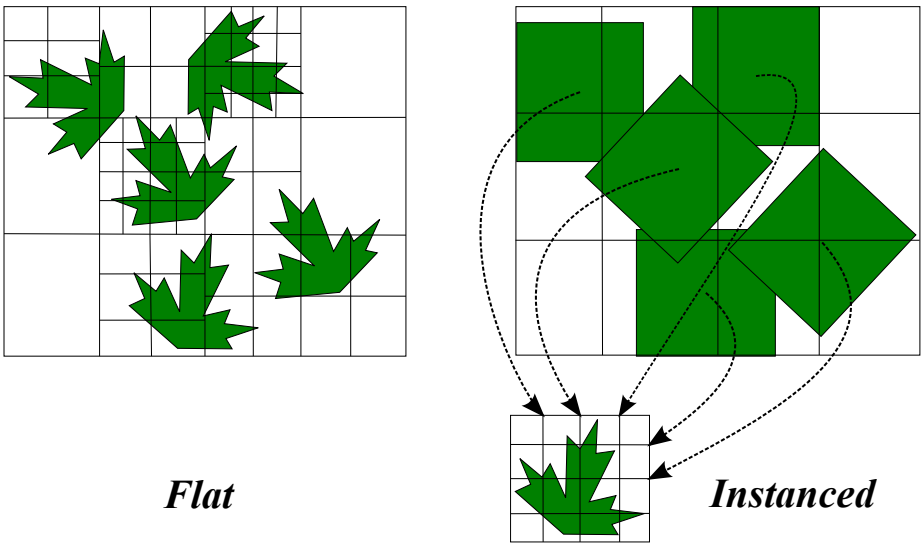


Figure 6.1: *Standard ray tracing acceleration structures are constructed over a flat scene representation (left). Geometry for every copy of the objects is stored separately in memory. Instancing enables to build a single spatial structure over the original instance and reference it multiple times in the top-level of the hierarchy (right).*

The ray tracing system we use in this chapter allows for supporting fully dynamic scenes, because we can rebuild both levels of the acceleration structure quickly. This is possible by using fast to construct uniform grids as an acceleration structure for both levels of the hierarchy and build them in parallel on a GPU. We generalize the two-level grid construction algorithm by Kalojanov et al. [KBS11]. There, the authors build a two-level hierarchical grid over a triangle soup. Their algorithm efficiently constructs each level of the acceleration structure in parallel, by reducing the process to a sorting problem. In our implementation, we extended this approach and used a uniform grid for the upper level of the structure and a uniform grid for each second-level object. The difference with [KBS11] is that we can place each of the second-level grids at an arbitrary location, instead of restricting them to be contained inside a single cell of the top-level grid (see Figure 6.1). This is done by storing independent bounding boxes for each second level element of the acceleration structure.

6.3 Inverse Instancing

So far in literature, instancing is always employed in a forward manner, meaning that in order to use it for a given scene, the objects and their copies have to be generated in advance. This limits the impact of the technique to scenarios where the consumer has the privilege of generating the geometric scene for rendering. In practice this often prevents the use of instanced geometry because geometric scenes are modeled as a triangle soup without explicit information about geometric similarity.

In this chapter, we will use the methods for microtile detection and tiling grammar simplification to compute a decomposition of the scene into reoccurring instances, such that the memory required to store the scene is minimal. In other words, we will use the microtile decomposition and grammar simplification methods to derive a symmetry-based method for loss-less triangle mesh compression. The theory we developed so far allows us to automatically decompose existing geometric scenes into reoccurring instances of a given type – the microtiles are instances of their respective classes. The algorithms for microtile detection can extract all parts of a geometric scene, that have been replicated.

Recall that the microtile decomposition minimizes redundancies in the geometric appearance of the input shape by identifying identical pieces of the surface. So far our experiments have shown that the unique pieces required to describe the input (and all of its similar shapes) compose only a small part of the complete input object. Therefore, a straight-forward application of the results in Chapter 3 and Chapter 4 can be used to compute a more-compact scene representation by representing the r-microtiles as instances.

6.4 A Cost Model for Mesh Compression

While it is clear that a microtile decomposition can be directly ray traced using the inverse instancing approach presented here, this is only useful, if the new representation is superior to the standard one in some aspect. Since instancing is primarily utilized to compact the memory footprint of the model, we will investigate the use of inverse instancing for triangle mesh compression.

So far we have shown that, using microtiles, one can minimize the redundancy in the geometric shape with respect to point-wise r-symmetry. Because the initial equivalence relations (partial symmetries) for the input mesh are defined per point, the measure of the redundant parts encoded in the classes of microtiles is zero. Therefore, the combined surface area of all microtile classes is minimal w.r.t. to partial symmetries.

The property of having minimal surface area is important for compressing the geometry. In many of our test models this representation also reduces the amount of triangles and storage necessary to represent the input scene for rendering. However, since the decomposition is triangulation independent and instancing an object introduces some memory overhead for storing the bounding box and inverse transformation matrix, the initial decomposition is not always optimal with respect to the memory footprint. In other words, while decomposing a shape into microtiles is triangulation invariant, the memory footprint of the microtiles still depends on their geometric representation. This means that instancing all r-symmetric parts might introduce memory overhead if the mesh triangulation does not allow to reduce the number of triangles necessary to describe the set of unique building blocks. This can be addressed by converting the initial decomposition into one with minimal memory footprint using the simplification method in Chapter 5.

6.4.1 General Cost Model

We will apply the grammar simplification algorithm using a cost function that characterizes the memory required to store the building blocks. In the case of triangle meshes the memory footprint depends on the triangulation of the particular mesh and consecutively the triangulation of each of the microtile classes. More precisely, the graph cost is

$$TotalCost := |TileInstances| \cdot C_I + \sum_{A \in TileClasses} \Delta_A, \tag{6.3}$$

where Δ_A denotes the space required to store all triangles in A, and as in the previous case, $|TileInstances|$ is the total number of tile instances and C_I is the cost for storing a single instance in memory. Here collapsing an edge between two nodes a, b of types A, B amounts to

$$Cost(a,b) := \begin{cases} \Delta_{AB} - \Delta_A - & \text{if } \forall As \text{ have adjacent } Bs \\ -\Delta_B - |I_B| \cdot C_I & \text{and } \forall Bs \text{ have adjacent } As, \\ \Delta_{AB} - \Delta_B - |I_B| \cdot C_I & \text{if } \forall Bs \text{ have adjacent } As, \\ \Delta_{AB} - \Delta_A - |I_A| \cdot C_I & \text{if } \forall As \text{ have adjacent } Bs, \\ \Delta_{AB} - |I_{AB}| \cdot C_I & \text{otherwise.} \end{cases}$$

Here $|I_A|, |I_B|$, and $|I_{AB}|$ denote the number of instances equivalent to a, b, and ab respectively, and Δ_{AB} is the storage for the geometry in the newly introduced building block consisting of a attached to b. Note that it is not possible to have the first case in the initial microtile graph because it will contradict the existence of different partial symmetries across the border of each 2 microtiles. It is however possible to arrive at a situation where the first case occurs after performing some edge collapses.

The second simplification operation: tile replacement always reduces the memory footprint because it simplifies the representation without introducing overhead. Therefore, we perform tile replacement whenever possible. However, since the opportunities for approximately matching large aggregated building blocks are usually rare, successful edge collapse operation have larger impact on the final cost.

6.4.2 Actual Memory Footprint

The reduction of the overall memory footprint is implementation specific and depends on the actual representation of tile classes and their instances, as well as the format used to represent the input triangles.

Triangles: In our evaluation, we assume a scenario in which the size of the scene is the main limitation. In this case, the input triangles are most compactly

represented as a triple of indices pointing inside a global vertex buffer, without additional pre-computed structure for faster ray-triangle intersections. In practice, the actual scene is more likely to be stored less efficiently. Often, in order to avoid the additional memory indirection, the triangles are stored as vertex triples, which requires to store multiple copies of each vertex in the scene. It is also common to pre-compute and store additional data structures that are used instead of the vertex coordinates during ray-triangle intersection tests. We measure the quality of our compression method as the reduction of the memory footprint compared to the initial representation. Therefore, the improvements we report are a worst case with respect to the most commonly used memory strategies for ray tracing triangle meshes.

Instances: The second implementation specific aspect of the evaluation of our method is the representation of the instances and the unique objects in the scene. These are accounted for in the variable C_I in Equation 6.3.

For each instance, we need to store a pointer (index) to the object it copies, and a rigid transformation that "transports" the object to the location of the instance. A naïve representation of the latter as a 3×3 rotation matrix and a translation vector is not very efficient, consuming 12 32bit floats. In our evaluation we reduced the necessary space by half using a translation vector and a normalized quaternion, which represents the rotational part of the transformation. We also hide a bit-flag in the object pointer, which we use to indicate whether or not the transformed instance had to be mirrored. This is necessary, because a combination of a rotation and a reflection cannot be represented by a single quaternion.

In our evaluation, the size of each instance is 28 bytes. We use this representation to evaluate our method, but we also implemented a 52 byte version, which stores a transformed bounding box, as well as a 76 byte variant with a complete transformation matrix. The larger variants allow for faster build times and better ray tracing performance. The differences in the compression rates for the three variants are minimal, which indicates that the overhead for instancing was not a major bottleneck for our method.

Additional overhead: When reporting our results, we take into account the memory overhead of the second level of acceleration structures constructed over the original exemplars. We use a standard two-level hierarchical grid as a baseline (see Figure 6.1 left, and [KBS11]) and count the resolution, the bounding box, and the two additional pointers as extra cost per tile class (Δ_A in Equation 6.3).

Even though we based the evaluation on our ray tracer implementation, the use of the optimization algorithm for other rendering applications is straightforward – it only requires adapting the cost terms C_I and Δ_A, which can usually be accurately estimated in advance.

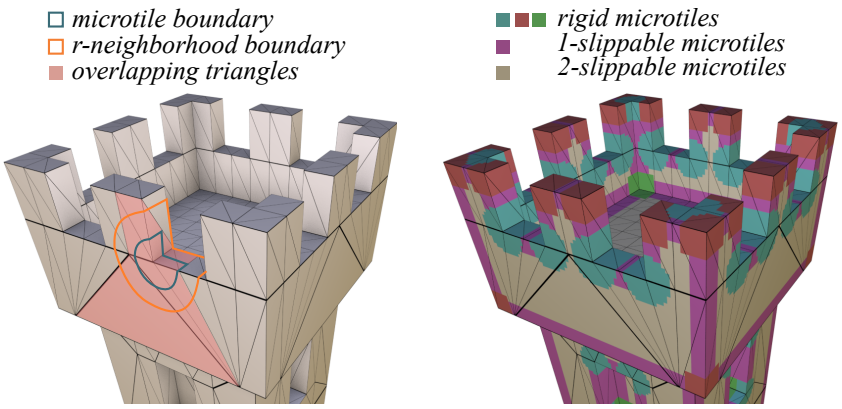


Figure 6.2: *The microtile decomposition is triangulation independent, meaning that the microtile boundaries do not coincide with triangle edges. Triangles protruding outside the r-neighborhood of a microtile cannot, in general, be replicated at each occurrence without causing visual artifacts.*

6.5 Surface Triangulation and r-Symmetry

In order to decompose the input triangle mesh into interchangeable surface segments, we have to consider that geometric correspondences we compute are triangulation independent as shown in Figure 6.2. If we naïvely select one microtile per class, find all triangles that it overlaps, and use them to represent the remaining equivalent microtiles as copies, we are likely to create visual artifacts like those in Figure 6.3. These are caused by triangles that extrude beyond the r-neighborhood of a microtile. Our symmetry model ensures equivalence of all surface points closer than the symmetry radius r to a microtile, i.e. only points on the input surface inside the r-neighborhood of a microtile can be safely replicated.

We therefore need to address triangles that span across the boundaries of the r-neighborhood of each microtile we chose as representative for its class. We developed two approaches that can be used to solve the problem. The first keeps the geometry unchanged and modifies the acceleration structure used to store it, erasing the excess parts. This is a convenient solution for a ray tracing system because there using a spatial index structure is necessary. The disadvantage of this approach is that the cells (or nodes) containing invalid geometry have to be re-computed or stored. We address this with a second method that replaces triangles extruding outside microtile neighborhoods with smaller ones. This introduces a small overhead and some additional triangles, but allows for discarding the r-symmetry information and storing the resulting triangle meshes in files using a slightly modified standard format.

Voxel Trimming: In our implementation, we were able to remove the visual artifacts like those in Figure 6.3 without modifying the input geometry. This is possible if we modify the acceleration structure storing the original copy. Because we used uniform grids in the second level of our hierarchy, we could build

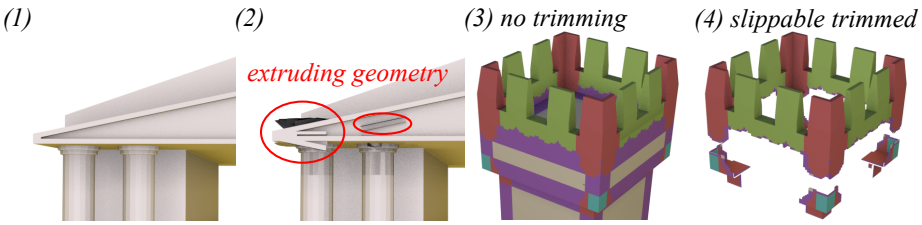


Figure 6.3: *(1) The upper corner of the courthouse model. (2) Naïve replication of the transformed instances can lead to artifacts caused by triangles on the microtile border that are only partially contained inside the r-neighborhood of the original. Ray tracing a model before (3) and after (4) marking as empty all cells of the acceleration structure containing geometry outside rigid microtiles. This technique resolves the artifacts seen in (2).*

the acceleration structure using cell sizes similar to the voxel size during symmetry detection. After the instances are passed to our renderer and we compute the uniform grids that contain them, we post-process the acceleration structures and "delete" cells outside the r-neighborhood of the tile see Figure 6.3 (3), (4). This means that we mark any cell outside the r-neighborhood of the tile as empty and do not render its contents. This approach might fail when using very coarse discretization, however the default voxel sizes we used for symmetry detection ($\frac{1}{256}$ and $\frac{1}{512}$ of the input scene diagonal) were small enough to allow for artifact-free rendering. In general, a voxel size smaller than $\frac{r}{2}$ allows for isolating the parts of the input surface inside the r-neighborhood of a microtile. The resulting surface will be identical to the non-instanced one, even if the actual triangles used to render the compressed version are too large.

Two-Slippable Pieces: The above voxel trimming method together with our decomposition approach present an additional opportunity for compression. As a part of the analysis we compute all planar regions of the mesh that have an area larger than πr^2. In some of our test models these regions were tesselated into multiple small triangles. We could replace these triangles by a single triangle or a quad covering the complete region. We shrink the bounding box of the triangle to the smallest one enclosing the original region. Similar to the non-slippable microtiles we modified the acceleration structure that contained them, by erasing the contents of the cells that were outside the original two-slippable region (see Figure 6.4. In other words we replaced each planar region by instancing a single triangle.

In our implementation, the footprint of each slippable region is 156 bytes distributed as follows:

- 36 bytes for triangle vertices
- 12 bytes for triangle indices (or 24 if we use a triangulated quad)
- 28 bytes for the geometry instance
- between 40 and 80 bytes for the additional uniform grid[1]

[1]We report results using an 80 bytes structure. A different memory organization of the acceler-

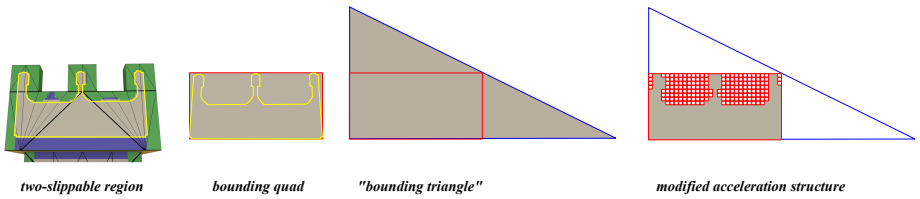


Figure 6.4: *We replace each two-slippable (planar) region with a single triangle that covers its bounding quad. We then modify the size and cells of the acceleration structure to "cut out" the extruding parts. We use this representation regardless of the underlying triangulation.*

For simplicity, we assume the same cell size of the acceleration structure for both the instanced version and the original model. This yields the same memory footprint of the cells and geometry references stored in both acceleration structures. This makes our representation more compact for any planar region that contains at least 6 triangles. These would occupy 168 bytes of space. This can be reduced to 4 triangles (120 bytes) by using a more compact uniform grid representation.

Note that fixing the cell size for rendering to the one used during symmetry detection limits the flexibility of the method: In a standard ray tracer the number of grid cells is set proportional to the number of triangles. Assuming standard grid density for ray tracing, i.e. around 5 cells per input triangle, we will introduce an initial memory overhead for meshes smaller than 3 million triangles. However, after reducing the model to instances, the acceleration structure built once per object type. The initial microtile decomposition usually eliminates more than 95% of the surface area for non-slippable geometry and we build acceleration structures over the remaining 5% of the mesh.

Boundary Re-Meshing: Using the acceleration structure to deal with excess geometry while keeping the local triangulation intact is a memory efficient solution, however it limits the method to ray tracing systems and requires to either store or re-compute the microtile boundaries before rendering the input mesh. An alternative solution, we implemented, is to replace triangles that extrude outside of the r-neighborhood of a microtile.

We consider 4 cases of triangles that are partially inside a microtile, based on the number of triangle vertices (0, 1, 2, or 3) within the microtile boundary. We use the strategy in Figure 6.5 for the cases with one or two vertices inside the boundary. Here, we assume that a triangle edge has at most one intersection with the microtile boundary. If we have 0 or 3 vertices, we find a vertex inside the triangle that is inside or outside respectively and recursively split the triangle into 3 new triangles that are likely to either one or two vertices inside and can be handled as in Figure 6.5. Triangles with edges that have multiple intersections with the microtile boundary require multiple levels of recursive subdivision.

Note that shortening the triangle edges as in Figure 6.5 can create holes if the

ation structure could use 36 bytes for grid resolution and 4 bytes for an index into a global storage for the grid cells.

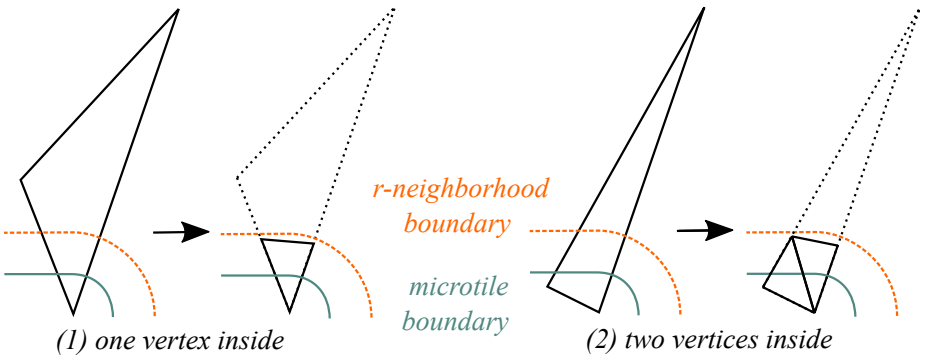
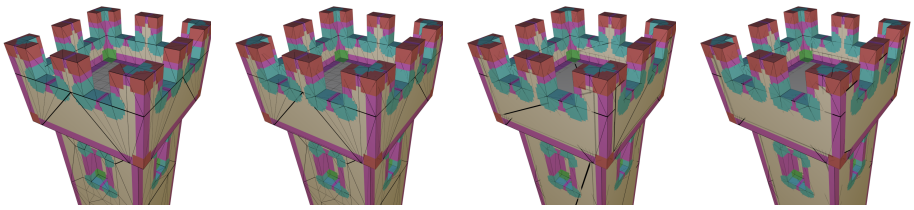


Figure 6.5: *We replace triangles with one or two vertices inside the microtile with smaller ones if they extrude beyond the r-neighborhood of the building block. Partially overlapping triangles with 0 or 3 vertices inside the boundary can be (recursively) split into triangles that have either 1 or 2 vertices inside the microtile.*

edge of a very long and thin triangle coincides with an edge feature, but is only partially contained inside the r-neighborhood of the microtile. This can be handled by re-meshing the input model in a pre-processing step, which might increase the memory footprint and negate any benefits from the subsequent compression. However such extreme cases are rare.

While this re-meshing method introduces additional triangles, the latter are considered when evaluating the cost function during optimization. This makes it possible to find a representation with minimal memory footprint, even if the latter is slightly larger than the one obtained with the voxel trimming method. The advantage of this approach is that it does not require special acceleration structure for rendering. This makes it possible to render the resulting compressed mesh using rasterization, and lifts the restrictions on the cell size for the acceleration structure for ray tracing. Further, it allows for storing the resulting instanced meshes using standard file formats that support instancing, e.g. VRML, X3D [Web]. We tested our method with a straight-forward extension of the Wavefront OBJ format [Wav].

In summary: We implemented two solutions that allow for inversely instantiating triangle meshes using r-symmetry regardless of the specific mesh triangulation (see Figure 6.6). In ray tracing applications, we modify the acceleration structure instead of the geometry to separate symmetric from non-symmetric parts of the model. This facilitates replacing large planar segments with just one or two triangles allowing to achieve better compression, especially for surface tessellated into large amounts of small co-planar triangles. Alternatively, we can modify triangles crossing microtile boundaries and export the resulting scene representation in files as well as render it using rasterization.



(1)
input microitles
(not instanced)

(2)
boundary re-meshing

(3)
voxel trimming

(4)
voxel trimming

Figure 6.6: *The initial triangle mesh (1) is decomposed into microtiles and then rendered using instancing. "Cutting out" building blocks from the mesh can be achieved by modifying either the geometry (2), or the acceleration structure (3) and (4). The latter allows for greater compression rates, but the results cannot be efficiently stored in files. We replaced each 2-slippable region with 2 triangles in (3), and with a single triangle in (4).*

6.6 Evaluation

We used the optimization algorithm from Chapter 5 for the Cost Function 6.3 and tested our implementation on various meshes. We initially expected that the reduction of the memory footprint will be related to the amount of partial symmetries discovered prior to the optimization step. Indeed, the number of possible representations through instances increases with the number of geometric redundancies in the input mesh. However, note that the overall compression rate also depends on how the individual building blocks are triangulated. This is the motivation for converting the initial triangulation independent redundancies into a representation that accounts for the actual memory footprint.

The compression results for the test models in Figures 6.7, 6.8, 6.9, and 6.10, vary significantly. While some models, like the "Companion Cube" (rightmost image in Figure 6.8 and 6.10, have partial symmetries that allow significant reduction in their memory footprint, simpler meshes like the asylum building in Figure 6.8 do not have complex non-slippable regions that can be compressed efficiently trough instancing.

The results in Figure 6.9 also show that the impact of the proposed techniques, e.g. replacing slippable geometry, depends on the use case. Certain models and the way they are triangulated lend themselves better to compression trough inverse instancing. Other models, might not have a triangulation and partial symmetries that allow a large improvement. On the upside, in most cases where compressing the model is not possible, given enough simplification steps, the optimization method will eventually merge all building blocks into a single one. This happens for example for the cathedral model in Figure 5.10.

One advantage of the method presented here is that it can handle non-manifold meshes with holes and other triangulation defects. This robustness is due to the combination of the approximate feature matching (see Section 4.4.2) combined with tile replacement (see Section 5.5). These techniques allow handling geometry with small deformation artifacts like the cube and the asylum building

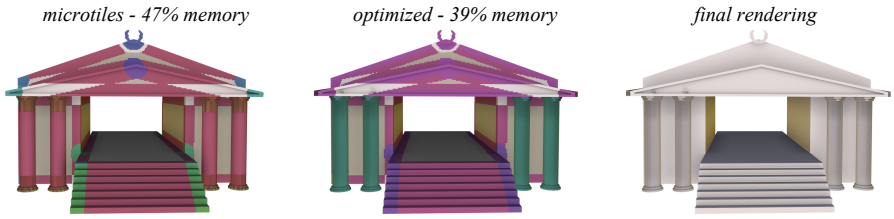


Figure 6.7: *The initial microtile decomposition for the courthouse reduces the footprint of the scene by 53% of its original size (left). After optimization, we further reduce the memory footprint by 61% (middle). Apart from some minor glitches due to self-occlusion, the rendering of the inversely instantiated scene (right) matches the original.*

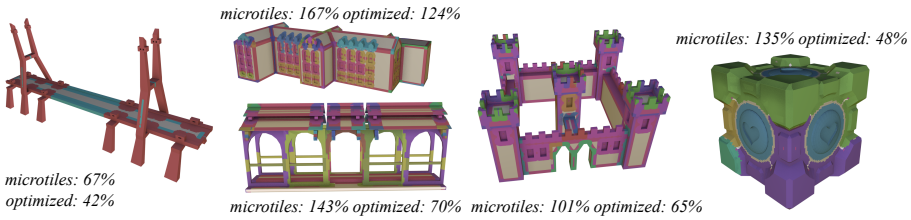


Figure 6.8: *Inversely instanced meshes ant their memory footprints relative to the size of the model before symmetry detection. The size of the initial decomposition into partial symmetries (microtiles) is reduced (optimized) using the grammar simplification method from Chapter 5. We used voxel trimming for all examples in the figure.*

in Figure 6.8.

In addition to the examples presented here, we used our ray tracing implementation to generate all rendered images in this thesis. All rendered scenes in this section show instantiated geometry.

6.7 Limitations

The compression approach we present has several limitations that affect rendering. Our method depends on the symmetry based decomposition, which is computed using a coarse voxelization of the input geometry. Because of that, we introduce overlapping geometry at the boundaries of neighboring geometry instances. Even though the radius of symmetry r allows for segmenting the models using approximate cuts, special care must be taken during rendering to prevent self-occlusion or self-shadowing, which can be observed for example on the pillars in Figure 6.7, right. Note however, that the color noise in the overlapping area on microtile borders (Figure 6.9 and 6.10), are only present, because we intentionally change the appearance of the different types of building blocks.

Further, the presented approach for mesh compression depends on the initial symmetry analysis, which limits the number of possible decomposition into instances. Therefore, the overall compression rate can be low if the input model

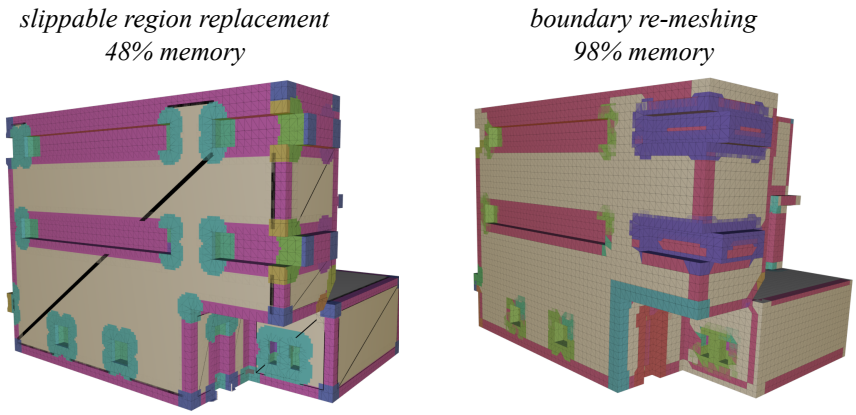


Figure 6.9: *Finely tessellated meshes are best compressed by replacing 2-slippable regions with proxy geometry. In this example, symmetry-based instancing of non-slippable geometry provides comparatively small amount of compression.*

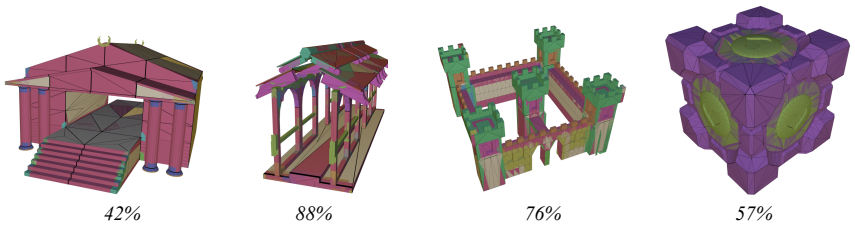


Figure 6.10: *Compression results using boundary re-meshing. While the compression rates are lower compared to the alternative method (Figure 6.8), the resulting models can be efficiently stored and transferred via files.*

only has a few geometric redundancies, or only few of these were successfully detected. A negative example presents itself in asylum building (Figure 6.8), where the symmetry structure and initial triangulation of the mesh do not provide a sufficient room for compression using inverse instancing resulting in a memory overhead compared to the non-instanced model.

6.8 Conclusion

In this chapter we applied a partial symmetry detection method for triangle mesh compression in the context of ray tracing. We used rigid r-symmetries and their induced microtiles to decompose triangle meshes into symmetric building blocks – a representation that minimizes geometric redundancy w.r.t. to rigid transformations of spherical point neighborhoods of a given radius r. We then used the tiling grammar simplification algorithm from the previous Chapter 5 and converted the initial decomposition into a set of triangle mesh excerpts and instances thereof, such that these minimize the memory footprint of the input model when rendered via ray tracing. We were able to obtain significant

reduction in size in some input models with suitable symmetry structure and triangulation. In addition, the compressed mesh representation can be ray traced directly without the need for decompression. Since the complete decomposition and optimization pipeline works without the need of user input or prior knowledge of the input model, we believe that the method presented here can be used in a number of ray tracing applications with limited memory footprint of the input scene.

7 | Building Shapes from Symmetries

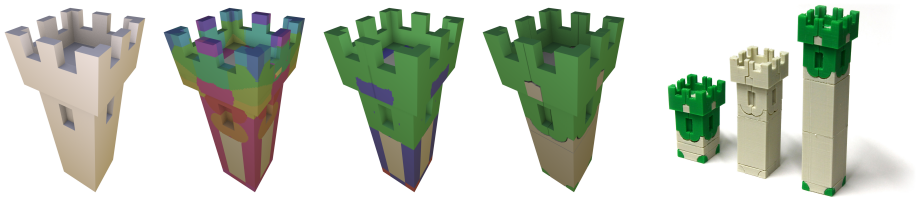


Figure 7.1: *Given a geometric model, we compute elementary building blocks from its partial symmetries, optimize the segmentation, and convert it to pieces that can be manufactured and assembled to produce shape variations.*

In this chapter, we use the microtile ability to describe shape families and develop an approach for shape decomposition into manufacturable building blocks that can be used to assemble the original shape as well as multiple variations of it. More specifically, we address the problem of finding a small amount of rigid pieces out of which an example 3D model as well as a large set of related shapes can be physically assembled.

In order to reduce the complexity and variety of microtiles, we apply the tiling grammar simplification approach introduced in Chapter 5. We formulate a cost function that quantifies the most important properties of the building blocks with respect to their 3D realization and mass-production via manufacturing processes like injection molding. In particular, we address the simplicity of the decomposition expressed in the number of types of pieces, the amount of geometric redundancies represented by the building blocks, the assemblability of the pieces and the amount of shape variation in the resulting shapes.

We demonstrate that the results of our algorithm can be used as building blocks in combination with 3D manufacturing techniques, where production of the pieces with the same shape is efficient. We also validate the functionality of the building blocks and their ability to assemble into multiple variations of the original shape, by printing prototypes with a common Fused Decomposition Modeling (FDM) 3D-printer.

7.1 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. 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.

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. Instead, 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.

Luo et al. [LBRM12] explores algorithms for shape partition. The difference between our work and theirs is that their primary objective is to reduce the size of the individual pieces and make them fit the maximum build volume of a given 3D printer. Although our method might potentially be used for similar purposes, we do not explore this objective here.

7.2 A Cost Model for 3D Manufacturing

Given a shape and its microtiles with their microtile graph, we define a cost model to measure the quality of the decomposition and its structure. We are interested in finding an optimal segmentation for a given application, in this case 3D manufacturing. To this end, we define a graph cost that depends on the nodes of the graph and search for a set of graph transformations minimizing this cost.

In this chapter, we focus on minimizing the complexity of the decomposition by reducing the number of different pieces and make them easier to manufacture and assemble. To this end, we define several desirable properties of the decomposition and try to express them in simple to compute properties of the microtiles and the microtile graph. We obtain significant improvement of the tile graph quality for these aspects, which are generic, but also relevant to applications like manufacturing and procedural modeling.

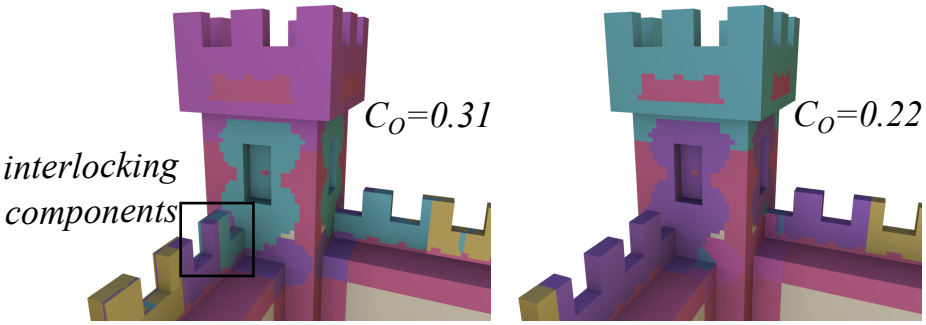


Figure 7.2: *The higher the overlap cost (C_O), the greater is the chance of producing interlocking components that can cause problems during assembly.*

Redundancy: One of the advantages of using corresponding building blocks is that they encode the redundancies in the shape geometry. The microtile decomposition is in fact optimal with respect to the r-similarity equivalence relation. Each different r-neighborhood is contained in exactly one microtile, hence the surface area covered by r-symmetric points represented more than once is zero (Lemma 7). We approximately compute the surface area covered by one instance per tile class i by counting the number n_i non-empty voxels intersected by the instance, as well as the total number of non-empty voxels in the scene N_V and define the cost

$$C_{SA} := \max\left\{\frac{\sum_i n_i}{N_V}, \alpha_{SA}\right\}, C_{SA} \in [0,1], \tag{7.1}$$

which approximates the fraction of non-redundant surface area in the representation. We limit the cost from below by 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]. Here, this translates to minimizing the number of tile classes. Intuitively, this reduces the production costs and minimizes the chances of producing defective elements or making assembly errors. 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 (also see Figure 5.3). Let $TC_{current}$ and $TC_{initial}$ be the set of microtile classes for the optimized and the initial decomposition. We define:

$$C_C := \frac{|TC_{current}|}{|TC_{initial}|}, C_C \in (0,1]. \tag{7.2}$$

Assemblability: Being able to ensure that once the 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 approaches (e.g. [XLF+11, SFCO12]) restrict the allowed shape of their puzzle pieces such that assemblability can be ensured trivially. We therefore use insights from the field of Mechanics and Design for Manufacturing and apply them to solve potential problems with the assembly of the manufactured pieces 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 number and size of the interfering volumes of neighboring pieces and hence minimizes the chance for interlocking sub-assemblies (see Figure 7.2). We define the cost for overlapping:

$$C_O = \frac{1}{N_T} \sum_{i=0}^{N_T} \frac{\sum_{j=0}^{N_T} \frac{V_{ij}}{V_i} \delta_{ij}}{\sum_{j=0}^{N_T} \delta_{ij}}, C_O \in [0,1], \tag{7.3}$$

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

$$\delta_{ij} := \begin{cases} 1 & \text{if tiles } i \text{ and } j \text{ are adjacent} \\ 0 & \text{otherwise.} \end{cases} \tag{7.4}$$

Note that the denominator $\sum_{j=0}^{N_T} \delta_{ij}$ is the number of neighbors of tile i, and is used to normalize the maximum value of the cost to 1.

Global Cuts: The building blocks we compute, represent all object variations constructable with non-slippable pieces. However some of the r-similar shapes require slippable building blocks with continuously varying size, which we cannot manufacture. In order to increase the shape variation we can actually build from the manufactured building blocks we define

$$C_{GC} := |n_{target} - n_{cuts}|, C_{GC} \in [0, \infty] \tag{7.5}$$

as the cost for the difference between a (user defined) desired (n_{target}) and the actual number (n_{cuts}) of symmetric global cuts. Each of these cuts splits regions of continuous symmetries into rigid parts that can be re-assembled into shape variations. We define and discuss how the cuts are computed in the following Section 7.3. It is important to note that some of the possible shape variations we obtain are due to these symmetric cuts.

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

$$Cost := \alpha_R C_R + C_C + C_O + C_{GC} \tag{7.6}$$

The weight α_R and the target number of global cuts can be used to control the amount of simplification during optimization. If not said otherwise, we set

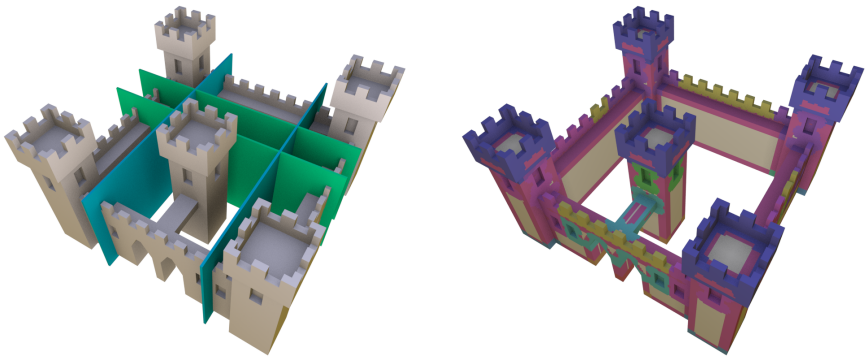


Figure 7.3: *An example model and two pairs of symmetric planar cuts (left) and an optimized microtile decomposition that preserves these cuts (right). We do not merge microtiles across cuts of this type to increase the number of shape variants we can construct.*

$\alpha_R = 1$ and vary only the number of global cuts in the remaining experiments. Scaling the term C_O in order to prevent complicated assembly is also possible, but was not necessary for our test models.

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.

7.3 Symmetry Cuts

How can we quantify variability? Unfortunately, tiling grammars are known to be able to encode the computation of 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 (see Figure 7.3). The same idea can also be adapted to general discrete graphs [LVW+15], then replacing r-symmetry with graph matching. We could efficiently compute a subset of these global cuts that subdivide slippable regions in addition to rigid microtiles.

Before the graph optimization step, we set a desired number of such global cuts that the final graph should contain (we used 2 or 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 7.5). If the number of remaining global pairs of cuts becomes less or equal to the target quantity, we

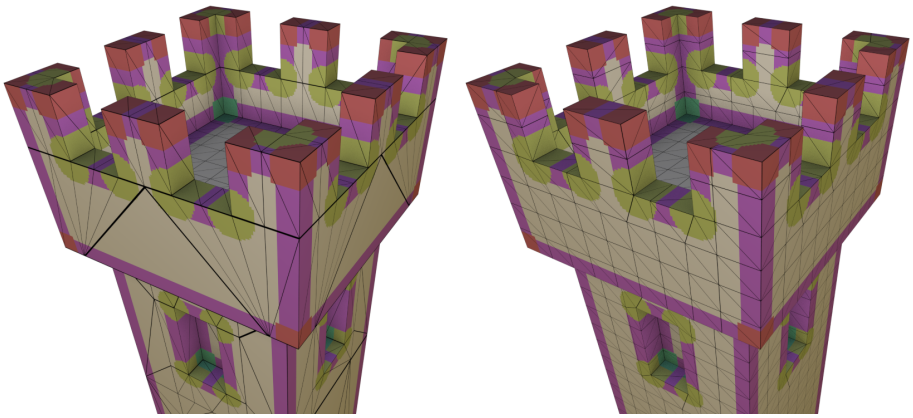


Figure 7.4: *The microtile decomposition is triangulation independent - two triangulations of the same surface yield the same segmentation. This implies that the microtile boundaries do not coincide with triangle edges. The building blocks are visualized via a voxel grid used as 3D texture. Rectilinear voxels can be used to color the surface, but do not yield pieces invariant under rotations.*

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 per-voxel because they can have arbitrary complicated shapes. For simplicity, we restrict the types of performed cuts when generating the geometry that has to be manufactured and compute all global **planar** cuts that intersect non-slippable edges through the middle – we separate rigid tiles from each other through the middle of the connecting edges. We also sample all slippable edges and find cuts that are contained entirely in slippable regions. This limits the types of cuts we can detect and we can only split the input shape with infinite planes (see Figure 7.3). On the other hand introducing this restriction speeds-up the search for global cuts, and allow for the cuts to be easily realized with simple CSG operations later on.

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. In the further process, this also splits intersecting planar (2-slippable) nodes.

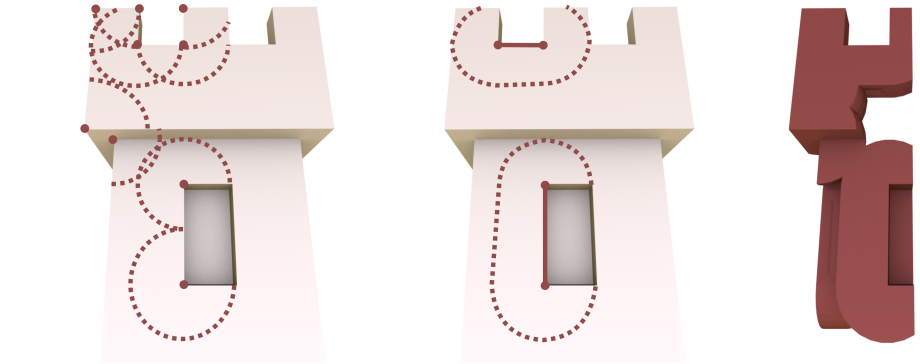


Figure 7.5: *An example collection of r-neighborhoods of corner features (left) and line features (middle). The resulting building block (right) is the result of a CSG intersection of the input model and the union of the spheres and cylinders enclosing the r-neighborhoods of the feature points and lines.*

7.4 Exact Microtiles

Our analysis so far is performed on a triangle mesh representing an input surface. In this section we describe 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 7.4, 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 manifests during assembly, when only a correct variant of a building block will fit given configuration of adjacent pieces.

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, and the possible graph simplifications that we use to improve the quality of the decomposition w.r.t. manufacturing.

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 already showed in Section 4.4.1, Lemma 6 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. Because each point on the triangle mesh is either rigid (non-slippable), one- or two-slippable, its r-neighborhood will either be planar, or it will contain a corner or edge. Therefore, the point

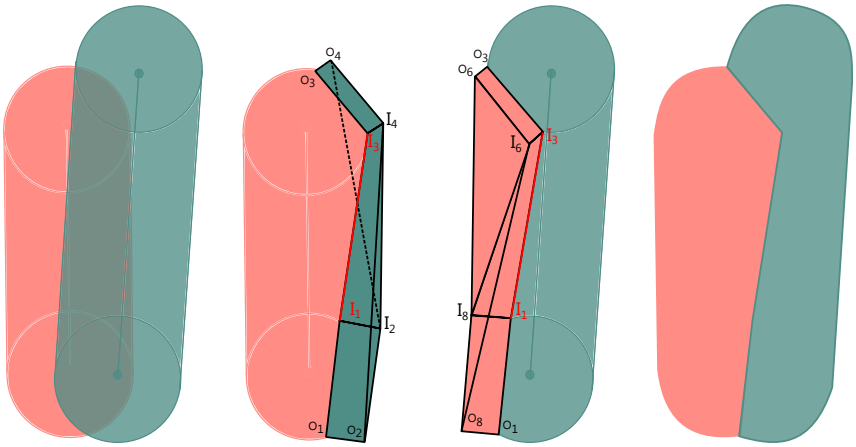


Figure 7.6: *Our naïve strategy for separating overlapping r-neighborhoods in 2D. From left to right – the initial situation, the subtracted polygons (second, third image), and the resulting cut. The quads (boxes in 3D) between inner (I_s) and outer (O_t) vertices are always subtracted, while the triangles $O_pI_qI_r$ and quads $I_1I_3I_qI_r$ $(p, q, r \in \{2, 4, 6, 8\})$ are not subtracted if any of the sides O_pI_q, O_pI_r, or I_qI_r is behind I_1I_3.*

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. See Section 4.4.1 for details.

The above considerations imply that the exact representation of each building block is contained inside the r-neighborhoods of a finite number of geometric features. These neighborhoods are easily enclosed by a set of spheres around corner features and cylinders along line features (see Figure 7.5). It is therefore possible to extract a 3D manufacturable representation of our r-symmetric building blocks via CSG operations with simple geometric primitives.

We implemented an algorithm that computes a CSG tree that encloses the r-neighborhood of each building block. Intersecting the tree with the input model delivers 3D manufacturable building blocks that can be assembled in the original shape and its variations. In order to verify our extraction method, we used a Fused Decompostion Modeling (FDM) 3D-printer to produce a set of constructor pieces for several test models.We now discuss how this is performed for the two types of building blocks we have: non-slippable (rigid) and (one- or two-) slippable.

7.4.1 Rigid Pieces

Each non-slippable building block in our segmentation is defined by a set of corner points, with r-neighborhoods matching the r-neighborhoods of all equivalent microtiles. Hence, the r-neighborhood of each microtile is contained inside a set of spheres (around corner features) and cylinders (along line features) by

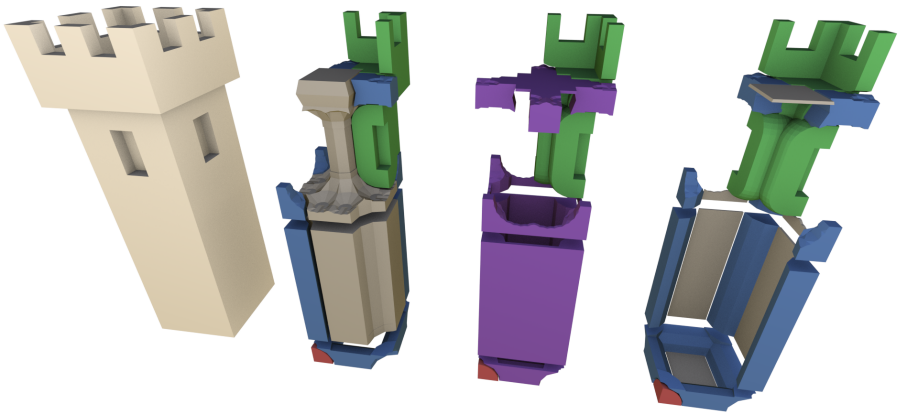


Figure 7.7: *The input (first model) can be decomposed into building blocks in three different modes. One-slippable and two-slippabe microtiles can be generated separately (blue and gray) or merged together (violet). It is sometimes possible to leave the two-slippable pieces attached to the enclosed volume (second model). Some pieces are removed for clarity.*

definition. This allows us to use a union of elementary shapes to cut out the microtile together with its r-neighborhood from the input shape using CSG operations. Similarly, by definition of r-symmetry each surface point at distance not greater than r to any geometric edge with both ends inside the microtile has an equivalent surface point in all microtile instances. Therefore, the geometry inside these volumes is guaranteed to be identical (equivalent) for all microtiles of a given type.

We need to take special care for overlapping r-neighborhoods of neighboring tiles when extracting building blocks that need to be put together after manufacturing. Intersecting r-neighborhoods of instances of the same tile class are present even in the simple example in Figure 7.5.

To resolve overlapping neighborhoods of tiles we modify the equivalence relation that defines our elementary building blocks. Initially microtiles are classified according to r-neighborhoods of corner features. We construct a new decomposition by splitting classes of microtiles if they have different neighbors at distance less or equal than $2r$. This enables classifying r-neighborhoods of corner points as well as intersections of these neighborhoods into equivalence classes.

At this point, it remains to consistently separate pairs of corners and edges with overlapping r-neighborhoods at each tile boundary. A general solution to this problem is to construct a 3D Voronoi diagram and split along its edges. For our tests, we constructed Voronoi regions only at corner points (see Figure 7.6). We then extended the regions along edges using the convex hull of the Voronoi regions at both end points of each edge. If pairs of neighborhoods constructed in this way had intersections, we assigned the intersecting volume to the smaller

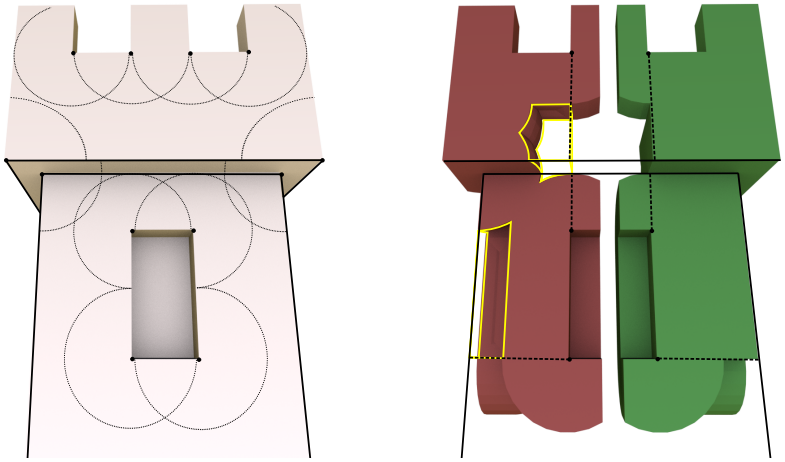


Figure 7.8: *Left – the input model (white), some of its corner features, their r-neighborhoods and all visible r-slippable edges (black). Right - we extend the r-neighborhoods of rigid pieces (green) along outgoing long edges to avoid concavities (outlined in yellow) that are likely to complicate the assembly of the manufactured pieces.*

neighborhood. It is important to note that using Voronoi regions to separate r-neighborhoods of corner points allows for preserving all planar cuts that split rigid geometry through the middle of geometric edges shorter than r.

Although not essential for the evaluation of our method, this ensures a canonical representation of the final pieces and allows for simplified extraction. This is the case, because the shape of the extracted building blocks remains the same, regardless of the order in which the pieces are ”cut out” of the input model.

Furthermore, for practical considerations, we extend the r-neighborhoods of each rigid (non-slippable) building block in two ways depicted in Figures 7.8 and 7.9. In both cases we merge parts of the splippable geometry together with rigid building blocks to make the assembly of the pieces less complicated. One-slippable edges are shortened whenever their enclosing cylinder intersect neighborhoods of rigid tiles at their end-points (see Figure 7.8). One-slippable and two-slippable regions contained completely inside a rigid building block are merged with the respective tile (see Figure 7.9). Both operations reduce the interfering volumes between slippable and non-slippable building blocks and may resolve issues with assemblability due to interlocking components. However, the main reason for implementing these aspects was to make assembly more convenient and reduce the concavities in the final pieces.

7.4.2 Slippable Pieces and Symmetry Cuts

After we have removed all rigid r-neighborhoods in the previous step, we can extract one-slippable tiles by intersecting the modified input object with a set of their enclosing cylinders.

The difference of the input shape and the remaining microtiles consists of all

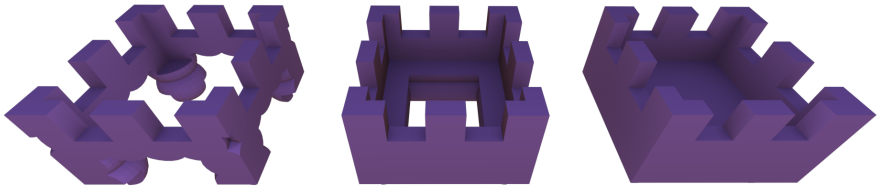


Figure 7.9: *The spherical r-neighborhoods of corners (left) are merged with one-slippable edges (middle) and two-slippable planar areas (right) of the model and then carved out of the input model using CSG operations.*

two-slippable building blocks attached to the volume enclosed by the shape. In order to separate the individual planar pieces from the inner volume, one can intersect them with their enclosing voxels. The last operation was not always necessary (see Figure 7.7), but it increases the likelihood that the pieces can be assembled after manufacturing. We parameterized the CSG tree for the slippable building blocks to allow three types of cutting as illustrated in Figure 7.7. For our experiments, we preferred to merge one- and two-slippable microtiles together. This resulted in fewer building blocks and did not prevent assembly for our test models.

As already discussed in Section 7.3, we separate slippable pieces along symmetric pairs of global planar cuts in order to increase shape variability. Each pair of planes defines an ”inner” and ”outer” region of the input model. From the intersections of these regions we obtain a general, not necessarily rectilinear, grid. We split the geometry in different grid cells by adding an intersection with the respective cell volume to the CSG-tree used to extract the final pieces.

Some pairs of symmetric planes isolate a region consisting entirely of slippable geometry, e.g. the middle section of the tower in Figure 7.7. If this is the case, valid shape variations can be obtained by ”stretching” or ”shrinking” the slippable building blocks. To construct variations of the same model in Figure 7.1 (right-most image) we computed segments with the original size and shorter ones , e.g. of length r.

7.5 Evaluation

We demonstrate that our approach is practical by implementing all stages of the pipeline depicted in Figure 7.1, apart from cutting the manufacturable building blocks form the original shape, for which we used OpenSCAD [Ope] – a tool for modeling CAD solids via scripts.

Decomposition Quality: We tested our method on several meshes of various complexity displayed in Figure 7.10, 7.11, 7.12 and 7.14. Note that we do not

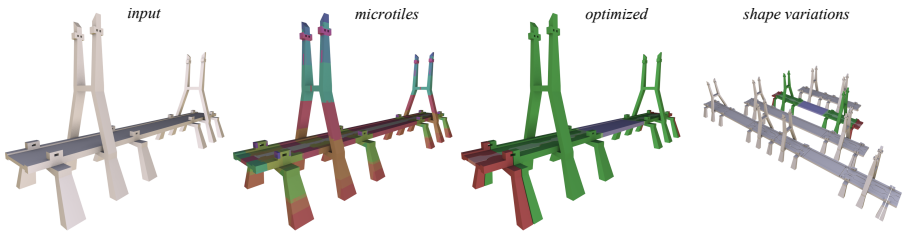


Figure 7.10: *A bridge is initially decomposed into 232 microtiles of 27 types, excluding slippable elements. The optimized decomposition with two global cuts consists of 13 pieces of 5 types including 5 slippable elements of 3 types. The initial cost of 1.39 was reduced to 0.26. We created some of the possible shape variations with the simplified building blocks.*

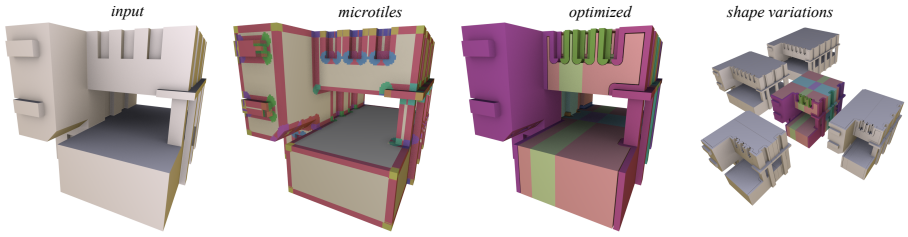


Figure 7.11: *A model of a modern building is initially decomposed into 246 microtiles of 36 types, excluding slippable elements. The optimized decomposition consists of 47 pieces of 28 types including 20 slippable elements of 17 types. The initial cost of 2.19 was reduced to 0.42. We preserved two global cuts and used them to construct some shape variations.*

refer to the triangle count, but the number of partial self-symmetries, when discussing the complexity of the input model. Our test models are subjects to self-symmetries generated by a large number (in the order of $10^4 - 10^5$) of transformations. Our test results indicate that the graph transformations allow for dramatically improving both the cost for the decomposition and the number of building blocks, without reducing the shape variability significantly (see Table 7.1 and Figures 7.10, 7.11, 7.12,7.14, and 7.13).

Another strong point of the optimized decompositions is the significant amount of redundancies encoded in the resulting pieces (measured by C_R) , which never represented more than 16% of all rigid neighborhoods. Even though we use abstract test criteria, the substantial improvements in the shape decomposition we obtain indicate the potential for a significant practical impact of our method.

Shape Variations: It is difficult to quantify by how much our simplifications reduce the variability of shapes, because characterizing all shapes that can be constructed from a context-sensitive tiling grammar is an undecidable problem. We can however, guarantee that a certain variability will be present, as long as there are some global symmetric cuts left in the decomposition.

For some models it makes sense to not compute the complete shape decomposition as in Figures 7.10, 7.11, and 7.12, because the resulting set of pieces might be too complex. In those cases it makes sense to modify the last step of the pipeline and generate constructor pieces by splitting the shape only along

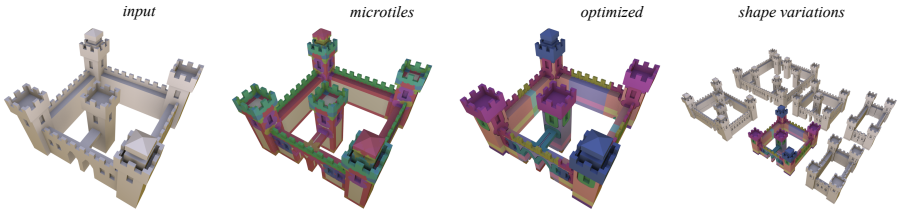


Figure 7.12: *A castle is initially decomposed into 693 microtiles of 53 types, excluding slippable elements. The optimized decomposition consists of 191 pieces of 39 types including 42 slippable elements of 27 types. The initial cost of 1.31 was reduced to 0.65. In addition to the three global cuts, we created shape variations by exchanging rigid pieces.*

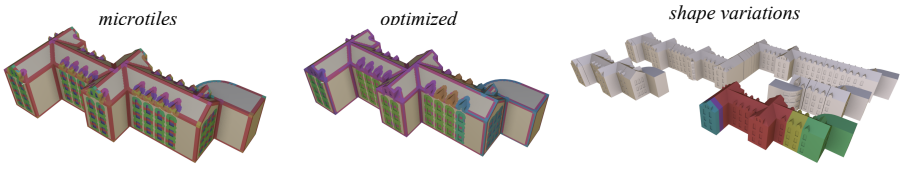


Figure 7.13: *An asylum building is initially decomposed into 956 microtiles of 206 types, excluding slippable elements. The optimized decomposition consists of 29 pieces of 13 types excluding slippable elements. The initial cost of 2.26 was reduced to 0.31. We only used the discovered global cuts to decompose the model and did not separate rigid and slippable microtiles.*

its global cuts (Figures 7.14, and 7.13). For each of these models, additional variations cannot be obtained by re-introducing the separation between rigid and slippable building block.

Note however, that in general the global cuts provide a subset of the shape variations. Individual rigid pieces can also be swapped, e.g. the tower tops in Figure 7.12 (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 7.1 and 7.10.

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 cannot formally guarantee the assemblability of the building blocks we compute, in practice, our heuristic works well: penalizing the overlap of the bounding volumes of neighboring pieces sufficed 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 7.15).

Complexity parameters: We also explore how we can adapt the granularity of our construction sets by varying parameters (Figure 7.17): Increasing the weight α_R for the redundancy term C_R in the cost formulation limits the amount of simplification and preserves more of the initially discovered partial symmetries. This increases the amount of shape variations. In Figure 7.17, the

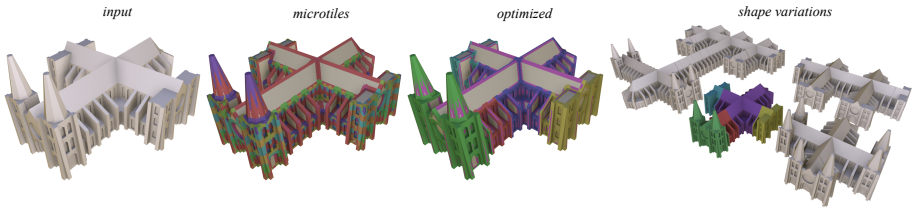


Figure 7.14: *A church is initially decomposed into 2276 microtiles of 395 types, excluding slippable elements. The optimized decomposition consists of 8 pieces of 5 types excluding slippable elements. The initial cost of 1.32 was reduced to 0.38. We only used the discovered global cuts to decompose the model and did not separate rigid and slippable microtiles.*

	# rigid pieces			# types of rigid pieces			cost		
	init.	opt.	diff.	init.	opt.	diff.	init.	opt.	diff.
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	2276	8	284×	395	5	79×	1.32	0.38	3×
asylum	956	29	32×	206	13	16×	2.26	0.31	7×

Table 7.1: *The improvement (difference) in the rigid building blocks and the overall cost of the decomposition we achieved after optimization.*

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 7.11 and 7.12. This is the most efficient stage in the pipeline – a naïve implementation would require hours for our test models, because of the tens of millions symmetry transformations candidates (see Chapter 4). The random search of a tile graph with optimal cost did take about 1 hour, because of the n^4 random samples of modified graphs we computed. 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 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

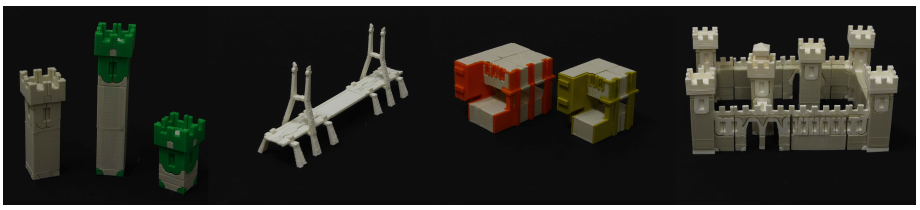


Figure 7.15: *Our test models assembled from 3D printed building blocks. We used a Makerbot Replicator 2 to print all models. We manually removed the pair of horizontal global cuts when manufacturing the castle model (right-most image) to simplify the assembly.*

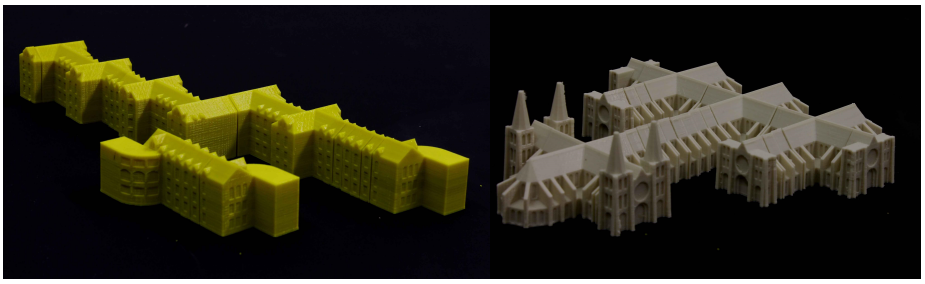


Figure 7.16: *Variations of the asylum and church models assembled from 3D printed building blocks. We used a Makerbot Replicator 2 to print all models and only split the input along global cuts to simplify assembly.*

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 7.12).

7.6 Limitations

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

While we are able to guarantee that the building blocks will fit together, we enforce assemblability with a soft constraint in our cost model. We penalize the interference between neighboring building blocks to reduce the chance of generating invalid assemblies. An interesting venue of future work is to find a less-conservative objective that can ensure the existence of a disassembly sequence for the final decomposition.

A property of the building blocks we neglected so far is the structural strength of the pieces. The scripts we use to generate the final pieces are parameterized and allow adjustment of the thickness of the slippable pieces for 3D printing, however the material and shape of the building blocks are currently not considered.

Finally, we 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 arbitrary shape is not easy to perform with the elementary CSG primitives and operations we used in our tests. An interesting continuation of this work would be to develop an approach to split the slippable parts of the input model using more general types of cuts.

7.7 Conclusion

In this chapter, we employ the theoretical and algorithmic insights from Chapters 3, 4, and 5 and make a first step towards developing a system for inverse procedural modeling of physically realizable shape collections. Even though some challenges remain to be addressed, our method already delivers promising results and provides means for automatic creation of 3D-printable constructor toys from virtual models.

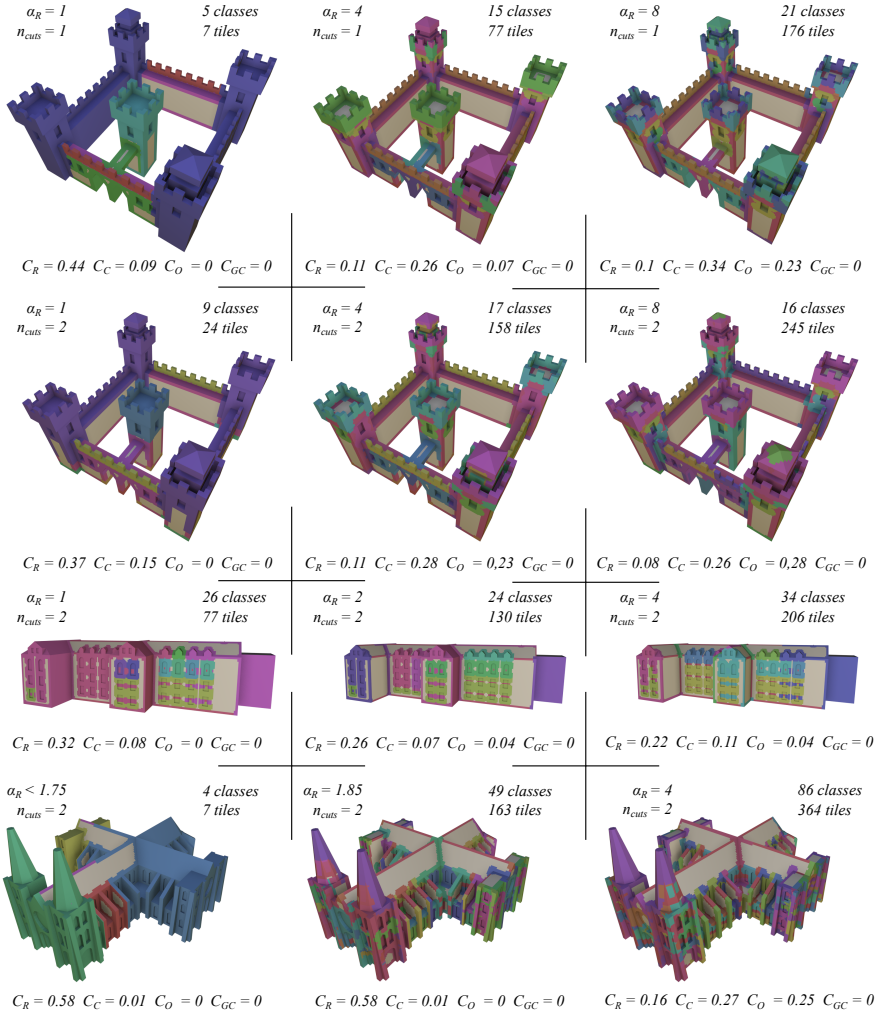


Figure 7.17: *The granularity of the building blocks can be controlled by weighting the redundancy term: C_R with α_R. Preserving more of the initially discovered symmetries can increase the amount of shape variations like the length of the castle walls or the possible facades of the asylum building. On the other hand, the smaller α_R the better the overall quality of the decomposition for manufacturing.*

8 | Conclusion

The goal of this work is to give insights into how shapes are structured from their partial symmetries. In the first part of this thesis we derive a universal definition of building blocks using point-wise self-correspondences.

While the basic concept of a microtile is quite abstract – a connected set of surface points that share their symmetry transformations, we show that these building blocks have a number of properties useful for characterizing shapes and for a specific type of correspondences (r-symmetry) collections of related shapes. The fundamental concept behind the definition of a microtile is to separate all parts of the shape "that are the same". In other words, trough the microtiles of a shape we structure its geometric redundancies and define a segmentation into sets of equivalent (or identical) pieces. The notion of equivalence encoded in the resulting decomposition allows to determine which parts of a model can be exchanged without visibly changing its geometric shape, and enables splitting the input into pieces that can be assembled in multiple ways, thereby creating shape variations.

The proof of Theorem 1 is the main theoretical contribution of this thesis. We show that using partial symmetries we can compute a formal description of an entire family of shapes. A key insight that we gained while working on the theoretical aspects of the microtile model, is that unlike global symmetries, which are characterized only by their inducing transformations, partial symmetries are defined by the corresponding pieces **and** the transformations inducing these correspondences.

A shape segmentation into building blocks, however interesting their properties, is not very useful if it cannot be computed efficiently. We address the extraction of microtiles with respect to rigid r-symmetry and develop an efficient algorithm for decomposing shapes into elementary exchangeable pieces. While the complexity of our algorithm is quadratic, we achieve runtimes fast enough to make microtile decomposition a practical geometry analysis tool.

Being able to quickly compute microtiles allows us to explore application scenarios for the two main concepts behind them – the geometric redundancies, and inverse modeling of similar shapes. However, it turns out that in both cases using partial symmetries as the only criteria might not suffice for a good segmentation. For example architectural models typically have a large number of self correspondences that generate a dense decomposition into small building blocks. This makes the building blocks unsuitable for mass production and manual assembly. We propose a solution with a graph-based tiling grammar simplification algorithm that optimizes the initial microtile decomposition for a particular application. The resulting segmentation still preserves some of the

initial symmetries, but trades some of the geometric redundancies for other important and application dependent properties. In other words, the graph-based simplification method allows for a good answer to a more specific question: "What are the best building blocks for application X?", as opposed to trying to find pieces that are optimal only with respect to partial symmetries.

We use the combination of symmetry based segmentation and subsequent simplification to develop a novel solution to a popular problem – triangle mesh compression. This application maps well to the ability of the microtiles to encode geometric redundancies. We can then convert the initial redundancies in the input shapes into redundancies w.r.t. the shape and its specific triangulation. An interesting aspect of our method is that using instancing, allows for ray tracing the compressed representation, without having to decompress in advance.

Finally, the we try to address the challenging problem of automatically computing physically realizable families of shapes. The unique properties of microtiles that relate them to tiling grammars for shapes, in addition to our grammar simplification method, allow us to make first steps towards automatically decomposing models into manufacturable sets of pieces that assemble into a number of shape variations. While our solution leaves a lot of avenues for future work, after addressing a number of technical challenges we are able to create constructor sets consisting of a few 3D manufacturable pieces that can be assembled into a large number of shape variants – the most significant result of this work.

With this work, we hope to have provided useful insights in the areas of symmetry-based shape analysis, shape understanding, and automatic shape creation. While some of the problems we approach are not fully solved, we believe that a symmetry-induced shape segmentation can have a variety of applications, and therefore the core concepts we explore in this work can prove useful. The practical methods in the the last two chapters provide evidence for this claim.

Bibliography

[AMWW87] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity, and symmetries of geometric objects. In *Proceedings of the Third Annual Symposium on Computational Geometry*, SCG '87, pages 308–315, New York, NY, USA, 1987. ACM. 4.2

[Ata85] M.J. Atallah. On symmetry detection. *IEEE Transactions on Computers*, 34(7):663–666, 1985. 4.2

[BAK10] Amit Bermanis, Amir Averbuch, and Yosi Keller. 3dd symmetry detection and analysis using the pseudo-polar fourier transform. *Int. J. Comput. Vision*, 90(2):166–182, November 2010. 4.2

[BBW+08] A. Berner, M. Bokeloh, M. Wand, A. Schilling, and H.-P. Seidel. A Graph-Based Approach to Symmetry Detection. In *IEEE/ EG Symposium on Volume and Point-Based Graphics*. The Eurographics Association, 2008. 4.2

[BBW+09a] Alexander Berner, Martin Bokeloh, Michael Wand, Andreas Schilling, and Hans-Peter Seidel. Generalized intrinsic symmetry detection. Research Report MPI-I-2009-4-005, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, August 2009. 4.2

[BBW+09b] M Bokeloh, A Berner, M Wand, H P Seidel, and A Schilling. Symmetry detection using feature lines. *Computer Graphics Forum*, 28(2):697–706, 2009. 3.1, 4.1, 4.2, 4.3, 3, 4.3.2, 4.2, 4.3.3, 4.4.2, 5.5

[BCBSG10] Mirela Ben-Chen, Adrian Butscher, Justin Solomon, and Leonidas Guibas. On discrete killing vector fields and patterns on surfaces. *Computer Graphics Forum*, 29(5):1701–1711, 2010. 4.2

[Ber66] Robert Berger. The undecidability of the domino problem. *Memoirs of the American Mathematical Society*, 66, 1966. 7.3

[BIT04] Pravin Bhat, Stephen Ingram, and Greg Turk. Geometric texture synthesis by example. In *Symp. Geometry Processing*, 2004. 3.1

[Boo96] Geoffrey Boothroyd. Design for manufacture and assembly: The boothroyd-dewhurst experience. In *Design for X*, pages 19–40. Springer Netherlands, 1996. 7.2

[BWKS11] Martin Bokeloh, Michael Wand, Vladlen Koltun, and Hans-Peter Seidel. Pattern-aware shape deformation using sliding dockers. *ACM Trans. Graph.*, 30:123:1–123:10, December 2011. 4.2, 4.3.2

[BWM+11] Alexander Berner, Michael Wand, Niloy J. Mitra, Daniel Mewes, and Hans-Peter Seidel. Shape analysis with subspace symmetries. *Computer Graphics Forum*, 30(2):277–286, 2011. 4.2

[BWS10] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph.*, 29:104:1–104:10, July 2010. 3.1, 3.3, 3.5, 4.1, 4.2, 4.3, 1, 4.3.2, 4.3.3, 4.4, 5.1, 7.3

[Cho59] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959. 3.5

[CK10] Michael Chertok and Yosi Keller. Spectral symmetry analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1227–1238, 2010. 4.2

[DA03] J Dong and G Arndt. 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):299–312, 2003. 7.1

[DMS06] Andreas Dietrich, Gerd Marmitt, and Philipp Slusallek. Terrain guided multi-level instancing of highly complex plant populations. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 169–176, September 2006. 6.2

[EKS+10] Michael Eigensatz, Martin Kilian, Alexander Schiftner, Niloy J. Mitra, Helmut Pottmann, and Mark Pauly. Paneling architectural freeform surfaces. *ACM Transactions on Graphics*, 29(4):45:1–45:10, 2010. 7.1

[EL99] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Int. Conf. Comp. Vision*, 1999. 3.1

[FLHCO10] Chi-Wing Fu, Chi-Fu Lai, Ying He, and Daniel Cohen-Or. K-set tilable surfaces. *ACM Trans. Graph.*, 29(4):44:1–44:6, July 2010. 7.1

[GCO06] R. Gal and D. Cohen-Or. Salient geometric features for partial shape matching and similarity. *ACM Trans. Graph.*, 25(1):130–150, 2006. 4.2

[GFW+06] Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek. Ray Tracing Animated Scenes using Motion Decomposition. *Computer Graphics Forum*, 25(3):517–525, 2006. (Proceedings of Eurographics). 6.2

[GG04] N. Gelfand and L. Guibas. Shape segmentation using local slippage analysis. In *Proc. Symp. Geom. Processing*, 2004. 3.3, 4.2, 4.3

[GYL+13] Jianwei Guo, Dong-Ming Yan, Er Li, Weiming Dong, Peter Wonka, and Xiaopeng Zhang. Illustrating the disassembly of 3d models. *Computers and Graphics*, 37(6):574 – 581, 2013. Shape Modeling International (SMI) Conference 2013. 7.1

[JTRS12] Arjun Jain, Thorsten Thormahlen, Tobias Ritschel, and Hans-Peter Seidel. Exploring shape variations by 3d-model decomposition and part-based recombination. *Computer Graphics Forum*, 31(2pt3):631–640, 2012. 5.1

[Kal15] Javor Kalojanov. Efficient r-Symmetry detection for triangle meshes. Technical report, Saarland University, 2015. 1

[KBS11] Javor Kalojanov, Markus Billeter, and Philipp Slusallek. Two-Level Grids for Ray Tracing on GPUs. In *EG 2011 - Full Papers*, pages 307–314, Llandudno, UK, 2011. Eurographics Association. 6.2, 6.4.2

[KBW+12] Javor Kalojanov, Martin Bokeloh, Michael Wand, Leonidas Guibas, Hans-Peter Seidel, and Philipp Slusallek. Microtiles: Extracting building blocks from correspondences. *Computer Graphics Forum*, 31(5):1597–1606, 2012. 1, 3, 4, 4.1, 4.3, 4.4, 4.4.7, 5.1

[KCD+03] Michael Kazhdan, Bernard Chazelle, David Dobkin, Thomas Funkhouser, and Szymon Rusinkiewicz. A reflective symmetry descriptor for 3d models. *Algorithmica*, 38(1):201–225, October 2003. 4.2

[KFR04] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Symmetry descriptors and 3d shape matching. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pages 115–123, New York, NY, USA, 2004. ACM. 4.2

[KLCF10] Vladimir Kim, Yaron Lipman, Xiaobai Chen, and Thomas Funkhouser. Möbius transformations for global intrinsic symmetry analysis. *Computer Graphics Forum (Symposium on Geometry Processing)*, 29(5), jul 2010. 4.2

[KWS16] Javor Kalojanov, Michael Wand, and Philipp Slusallek. Building Construction Sets by Tiling Grammar Simplification. *Computer Graphics Forum*, 35(2):013–025, 2016. 1

[LBRM12] Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. Chopper: Partitioning models into 3D-printable parts. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 31(6), 2012. 7.1

[LCDF10] Yaron Lipman, Xiaobai Chen, Ingrid Daubechies, and Thomas Funkhouser. Symmetry factored embedding and distance. *ACM Transactions on Graphics (SIGGRAPH 2010)*, July 2010. 3.1, 4.1, 4.2, 4.3.2, 3

[Low99] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society. 4.4.2

[LTSW09] Ruxandra Lasowski, Art Tevs, Hans-Peter Seidel, and Michael Wand. A probabilistic framework for partial intrinsic symmetries in geometric data. In *IEEE International Conference on Computer Vision (ICCV'09)*, page toappear, Koyoto, Japan, 2009. IEEE Computer Society. 4.2

[LVW+15] Han Liu, Ulysse Vimont, Michael Wand, Marie-Paule Cani, Stefanie Hahmann, Damien Rohmer, and Niloy J. Mitra. Replaceable substructures for efficient part-based modeling. *Computer Graphics Forum*, 34(2):503–513, 2015. 5.1, 5.2, 5.8, 7.3

[Mak] MakerBot. http://www.makerbot.com. 7.5

[MBB10] Niloy J. Mitra, Alex Bronstein, and Michael Bronstein. Intrinsic regularity detection in 3d geometry. In *ECCV*, 2010. 3.1, 4.2

[Mer07] Paul Merrell. Example-based model synthesis. In *Symp. Interactive 3D Graphics and Games*, pages 105–112, 2007. 3.1

[MGP06] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph.*, 25(3):560–568, 2006. 3.1, 4.1, 4.2

[MPWC12] Niloy J. Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3d geometry: Extraction and applications. In *EUROGRAPHICS State-of-the-art Report*, 2012. 4.1, 4.2, 4.3

[MWZ+13] Niloy J. Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, and Martin Bokeloh. Structure-aware shape processing. In *EUROGRAPHICS State-of-the-art Report*, 2013. 5.1

[MYY+10] Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. Illustrating how mechanical assemblies work. *ACM Trans. Graph.*, 29(4):58:1–58:12, July 2010. 7.1

[Ope] OpenSCAD. http://www.openscad.org. 7.5

[OSG08] Maks Ovsjanikov, Jian Sun, and Leonidas Guibas. Global intrinsic symmetries of shapes. In *Eurographics Symposium on Geometry Processing (SGP)*, 2008. 4.2

[PMW+08] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. Guibas. Discovering structural regularity in 3D geometry. *ACM Trans. Graph.*, 27(3), 2008. 3.1, 4.2

[PSG+06] Joshua Podolak, Philip Shilane, Aleksey Golovinskiy, Szymon Rusinkiewicz, and Thomas Funkhouser. A planar-reflective symmetry transform for 3D shapes. *ACM Trans. Graph.*, 25(3), 2006. 4.2

[RBB+10] D. Raviv, A. M. Bronstein, M. M. Bronstein, R. Kimmel, and G. Sapiro. Diffusion symmetries of non-rigid shapes. In *Proc. 3D Data Processing, Visualization and Transmission (3DPVT)*, 2010. 4.2

[RBBK07] D. Raviv, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Symmetries of non-rigid shapes. In *Proc. Non-rigid Registration and Tracking (NRTL) workshop. See Proc. of International Conference on Computer Vision (ICCV)*, oct 2007. 4.2

[RBBK10] D. Raviv, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Full and partial symmetries of non-rigid shapes. *Intl. Journal of Computer Vision (IJCV)*, 89(1):18–39, August 2010. 4.2

[ŠBM+10] Ondrej Šťava, Bedrich Beneš, Radomir Měch, Daniel Aliaga, and Peter Krištof. Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum*, 2010. 5.1

[SFCO12] Peng Song, Chi-Wing Fu, and Daniel Cohen-Or. Recursive interlocking puzzles. *ACM Trans. Graph.*, 31(6):128:1–128:10, November 2012. 7.1, 7.2

[SG99] N. Shyamsundar and Rajit Gadh. Geometric abstractions to support disassembly analysis. *IIE Transactions*, 31(10):935–946, 1999. 7.1, 7.2

[SKS06] Patricio Simari, Evangelos Kalogerakis, and Karan Singh. Folding meshes: hierarchical mesh segmentation based on planar symmetry. In *Proc. Symp. Geometry Processing*, pages 111–119, 2006. 4.2

[SOG09] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum*, 28(5):1383–1392, 2009. 4.2

[SS64] W.R. Scott and W.R. Scott. *Group Theory*. Dover Books on Mathematics. Dover Publications, 1964. 2.1

[SS10] Mayank Singh and Scott Schaefer. Triangle surfaces with discrete equivalence classes. *ACM Trans. Graph.*, 29(4):46:1–46:7, July 2010. 7.1

[Thi] Thingiverse. http://www.thingiverse.com. 4.7

[TW05] Sebastian Thrun and Ben Wegbreit. Shape from symmetry. In *Proc. Int. Conf. Computer Vision*, 2005. 4.2

[TYK+12] Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 63–74. ACM, 2012. 5.1

[Wav] Wavefront Technologies. https://en.wikipedia.org/wiki/Wavefront_.obj_file. 6.5

[Web] Web3D Consortium. http://www.web3d.org/standards/all. 6.5

[WL00] L.Y. Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proc. Siggraph*, 2000. 3.1

[WYD+14] Fuzhang Wu, Dong-Ming Yan, Weiming Dong, Xiaopeng Zhang, and Peter Wonka. Inverse procedural modeling of facade layouts. *ACM Trans. Graph.*, 33(4):121:1–121:10, July 2014. 5.1

[XLF+11] Shiqing Xin, Chi-Fu Lai, Chi-Wing Fu, Tien-Tsin Wong, Ying He, and Daniel Cohen-Or. Making burr puzzles from 3d models. *ACM Trans. Graph.*, 30(4):97:1–97:8, July 2011. 7.1, 7.2

[XZT+09] Kai Xu, Hao Zhang, Andrea Tagliasacchi, Ligang Liu, Guo Li, Min Meng, and Yueshan Xiong. Partial intrinsic reflectional symmetry of 3d shapes. *ACM Trans. Graph.*, 28(5):138:1–138:10, December 2009. 4.2

[ZCOM13] Youyi Zheng, Daniel Cohen-Or, and Niloy J. Mitra. Smart variations: Functional substructures for part compatibility. *Computer Graphics Forum (Eurographics)*, 32(2pt2):195–204, 2013. 5.1

[ZHW+06] Kun Zhou, Xin Huang, Xi Wang, Yiying Tong, Mathieu Desbrun, and Heung-Yeung Shum. Baining Guo. Mesh quilting for geometric texture synthesis. *ACM Trans. Graph.*, 25(3):690–697, 2006. 3.1

[ZPA95] Hagit Zabrodsky, Shmuel Peleg, and David Avnir. Symmetry as a continuous feature. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(12):1154–1166, December 1995. 4.2