Technical Report, March 2015.

Efficient r-Symmetry Detection for Triangle Meshes

Javor Kalojanov*
Saarland University

Abstract

In their paper [2012], Kalojanov et al. introduce 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. The authors 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. Here, 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. Afterwards, we introduce an efficient method for computing microtiles, which permits such involved analysis to be used in variety of geometry processing applications.

Keywords: geometric modeling, symmetry detection, inverse procedural modeling, shape understanding

1 Introduction

Partial symmetry detection and its uses in the field of Computer Graphics has been studied extensively in the recent years, as surveyed by Mitra et al. [2012]. 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 [Mitra et al. 2006; Bokeloh et al. 2009; Lipman et al. 2010]. 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 by Kalojanov et al. [2012], where all point-wise equivalences must be detected 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 make the microtile analysis impractical.

In this document, we present two algorithms for microtile decomposition and r-symmetry detection. First, we derive a implementation using the work by Bokeloh et al. [2010] as a starting point and use it to verify the theoretical concept from Kalojanov et al. [2012] and the proof of main theorem in their paper. The first part of the document is briefly discussed by Kalojanov et al. [2012]. 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 the most substantial parts of it can be efficiently implemented using parallel architectures such as graphics cards. We are able to achieve decomposition times under a minute for meshes of non-trivial complexity, detecting hundreds of thousands of partial correspondences on a commodity PC.

*e-mail:kalojanov@cs.uni-saarland.de

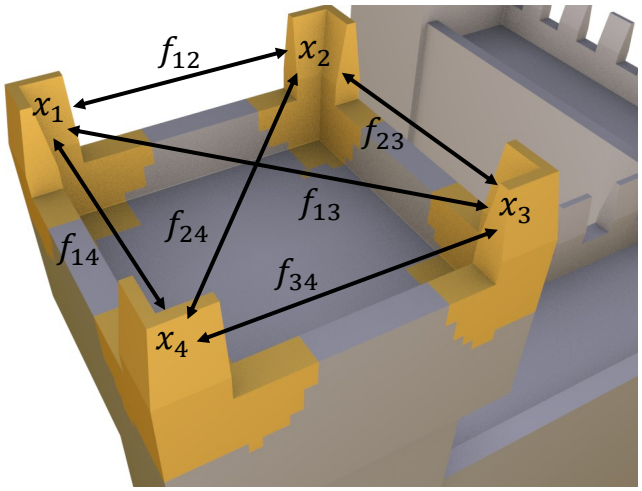

Figure 1: *A graph representation of an example set of partial mappings. The nodes are (colored) points on the surface and the edges are mappings. As illustrated, all connected components of the graph have to be cliques by definition.*

2 Related Work

The algorithms for symmetry detection we discuss are closely related to the works of Bokeloh et al. [2009; 2010]. There, the authors identify and match line features to detect similar or corresponding regions on the surface of the input model. Bokeloh et al. [2010] detect matching points that lie on pairs of r-symmetric cuts through the model. To decompose a shape into microtiles one need 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. [2010] 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 like triangle mesh compression and 3D manufacturing, discussed in several related works by Kalojanov et al.

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 [2012] 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. [2012], we proceed in three stages: **feature extraction**, **aggregation** and **extraction**.

Feature Selection: We cannot test infinitely many transformations as they appear in slippable regions. Therefore, we first perform slippage analysis for all r-neighborhoods [Gelfand and Guibas 2004]. Afterwards, we ignore slippable regions in the remaining computation. Then, we compute *line features* [Bokeloh et al. 2009]. 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 triangle fragments as representation [Bokeloh et al. 2010]). 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 1). We transform the voxels that belong to a given microtile, and search in the overlapping voxels for the equivalent microtile instance.

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. [2010].

2. We compute and mark all r-slippable voxels and treat them separately further on.

3. We then detect the line features that give queues for symmetry transformations s. [Bokeloh et al. 2009]. We use them to compute the actual set of candidate transformations that map pairs of r-symmetric points.

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.

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 [Bokeloh et al. 2010; Bokeloh et al. 2011; Lipman et al. 2010], 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 $\mathcal{S}$. 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 to decompose the 1-slippable microtiles. To this end, we need to compute all transformations that map pairs of line features to each other (s. [Bokeloh et al. 2009]).

It remains to find all transformations that map non-slippable neighborhoods symmetrically. Observe that any such region 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). 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.

3.3 Results

We have implemented a simple prototype of the algorithm outlined above. We follow the method of [Bokeloh et al. 2010] 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 2) 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). To prevent errors due to coarse discretization, classes of microtiles were computed only for microtiles larger than 32 voxels. Very small tiles

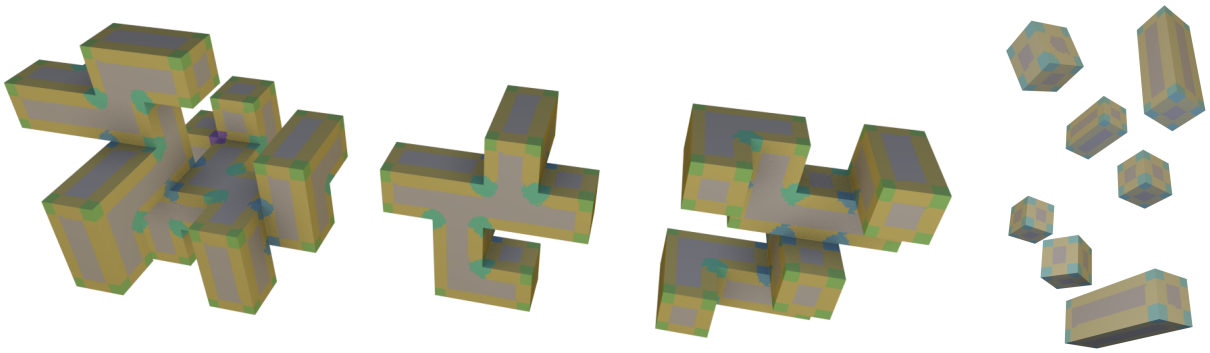

Figure 2: *Simple models. Left: Simultaneous decomposition of three simple meshes in one scene, $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$. Right: decomposition of 7 boxes in one scene. Equivalent tiles in each figure were computed and colored automatically. The number of different 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 decomposition of the first three meshes was around 10 min, the boxes on the right took approx. 1min.*

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 2 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 3, 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., oversegmentation of the steps of the staircase). Similar results are obtained for the models in Figure 4. We compare our results to the previous method by Bokeloh et al. [2009], which is computationally mostly similar but uses (as most others) simple region growing for segmentation. The method is similarly susceptible to discretization and boundary artifacts. It does not capture all symmetries, 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 2 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 3 is r-similar to the castle, which contains additional tiles. A further example is demonstrated in Figure 4 (right). A sequence of models with increasing complexity is decomposed into microtiles, revealing the redundancy in the model collection.

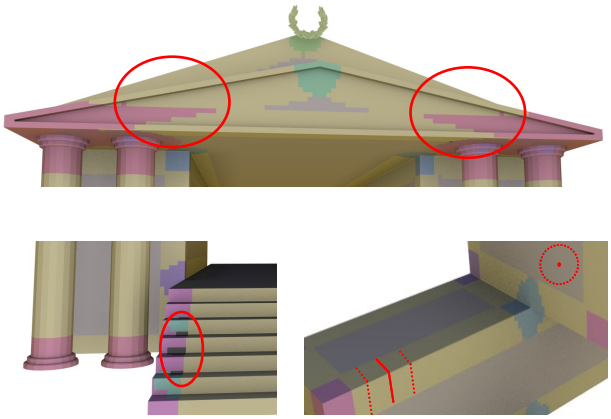

Figure 5: *Drawbacks of the naïve microtile extraction method: The global voxelization results in different voxel representations for equivalent microtiles (top) and artifacts inside voxels on tile boundaries (bottom left). One- and two-slippable microtiles are excluded from the analysis (bottom right).*

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 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 5). We are sometimes unable to correctly compute the equivalence classes or cliques inside voxesl on tile boundaries (see Figure 5). 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 be-

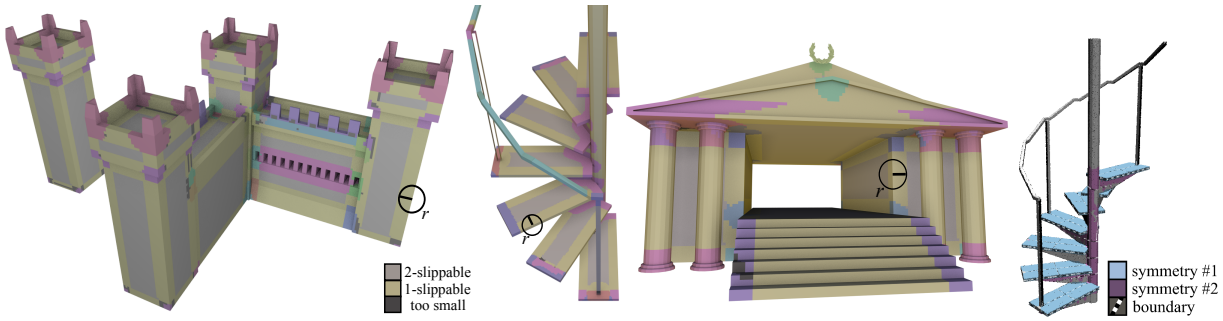

Figure 3: *More complex models: Castle (left), staircase (middle), courthouse (right) models decomposed into r-microtiles. The classes can be computed reliably only for microtiles larger than 32 voxels. The diameter of this uncertain area is smaller than r. Run-time: around 1 hour (castle), 18 min (staircase), and 40 min (courthouse). Far right: comparison to Bokeloh et al..*

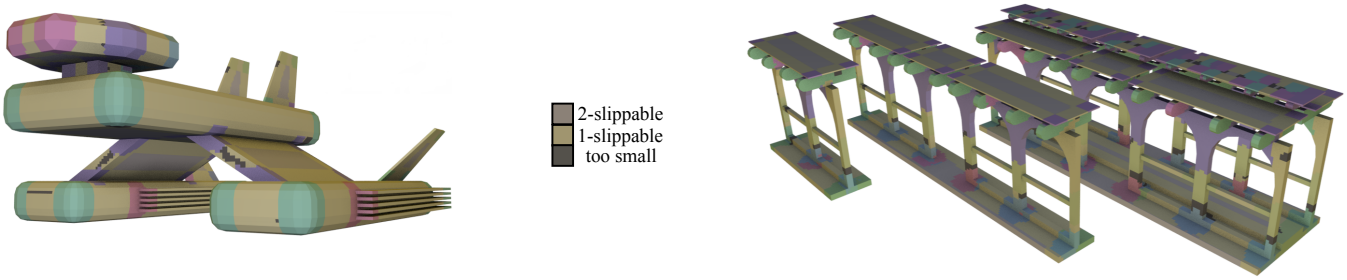

Figure 4: *A spaceship model (left) - the tiles are recognized correctly, but again, some unassigned area remains (gray). A cascade of models with increasing complexity (right) - the newly added parts create new tiles.*

comes 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 2,3). 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 and Figure 5 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 2, 3, 4).

4 Efficient r-Symmetry Detection

The microtile decomposition method used in [Kalojanov et al. 2012] 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, and 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 [Kalojanov et al. 2012; Bokeloh et al. 2010], the authors transform the complete scene and match it with itself using a spatial 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 naïve 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 to compute building blocks and cut the input surface exactly at the microtile boundaries, which is essential for applications involing the computation of 3D realizable building blocks.

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 perform the most time-consuming parts in parallel. On the other hand, we propose a different approach in terms of algorithm 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 representation 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 the r-neighborhoods 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 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 symmetrically (r-symmetrically).

Lemma 1 *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 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 $r \rightarrow 0$. It is then possible to deduce the mesh segmentation into microtiles from this intermediate representation.

Lemma 2 *It suffices to decompose the r-neighborhoods of all geometric corners and edges to obtain a surface decomposition into microtiles in the sense defined in Section 3 in [Kalojanov et al. 2012].*

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 does the corner feature 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 from [Kalojanov et al. 2012], because building blocks, which 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 the abstract definition of microtiles, presented in the related work, if we restrict the set of defining point-wise correspondences to the ones who 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 the main theorem in [Kalojanov et al. 2012] remains valid for the new decomposition. This follows from Lemma 1, which implies that all possible variations of rigid regions are represented by microtiles, which contain at least two non-parallel edges with distance smaller than r.

Benefits: The above lemma allows to simplify 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 need to compute all pairs of geometric corners and edges that are r-symmetric. If we match one-slippable regions separately, it is only necessary to match all geometric corners in addition to all r-neighborhoods defined by skewed geometric edges at distance shorter than r. In our experiments, we could safely ignore the latter, but they can be trivially handled by introducing a "virtual" corner feature for each such r-neighborhood.

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.

Restricting the decomposition to characteristic points has advantages not only in therms of the reduced running time of the algorithm. It also allows to exactly match 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.

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 in Figure 5.

Let $N_r(\mathbf{x})$ be a 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 1 *A **fragment** is a voxel-triangle overlap and consists of a position – the projection of the cell center onto the triangle and a normal – the geometric normal of the triangle (we do not consider fake or smooth normals). 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 2 *Let f, g be a 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° in our tests). Formally,*

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

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

Definition 3 *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 $(g.p, g.n)$.*

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

Definition 4 *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*

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

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.

Note that this formulation of similarity allows to match arbitrary sets of fragments, and is not restricted to individual r-neighborhoods. This allows to compare larger subsets of the input surface to each other and distribute 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.

Note that the approximate matching of r-neighborhoods of key-poins can be used to match noisy or deformed input data. However this does not allow to directly apply the algorithm on such models, because it is unclear how to select unique key-points to identify each non-slippable region of the input model. The benefits for our experiments on clean input models are the added robustness to numerical errors and the ability to match geometry undergoing very small deformations.

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. 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 a 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 r-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 6). We then compute all possible transformation 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 6).

   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 in 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 ($p \in [0.01, 0.04]$ in most of our experiments) to determine equivalence and convert each connected component to a clique by "inserting" the missing edges.

   An alternative approach for the same operation on noisy or deformed data can be implemented via spectral clustering similarly to Lipman et al. [2010].

4. Compute microtile membership for the corner features.

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

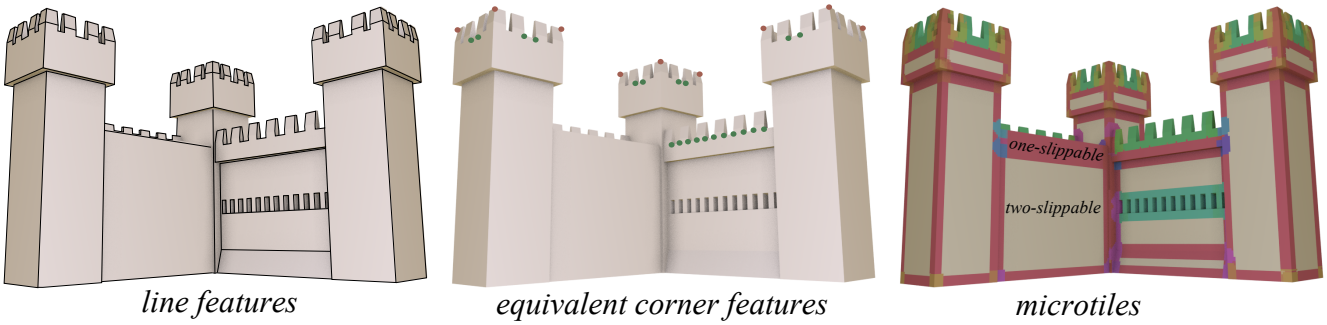

Figure 6: *Mircotile decomposition pipeline. First, all geometric edges are computed and later used to identify all corner features. Then r-neighborhoods of corner features are matched to compute equivalence cliques. Features and neighborhoods with the same outgoing equivalence transformations are merged 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.*

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 6).

   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 microtile decomposition.

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 to have one-one mappings 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 7).

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 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 point in the clique plus a single (and complete) set of transformations that maps one of the points to itself symmetrically. 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 addition to alleviating the need to store transformation matrices, this property of the microtiles allows to discard some of the candidate matches. We test and classify each corner feature according to the number of self-symmetries. When we are testing for equivalent corner features we only have to test a pair of them if they have the same number of self-symmetries. 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% test models like the courthouse in Figure 7 and the castle in Figure 6.

Another possibility to eliminate 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 7) 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 6 we could eliminate 12.39 million transformation candidates and only had to perform around 160000 matches.

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 1 and Lemma 2 allows us to use local matches instead. We compare the geometry inside (multiple)

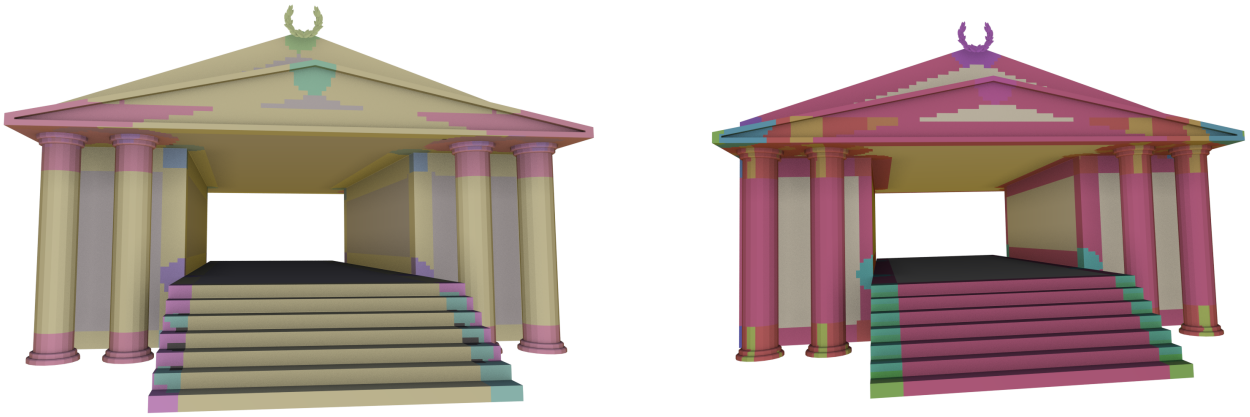

Figure 7: *Left: Output of the naïve decomposition algorithm for the courthouse model. The discretization artifacts on the stairs remain even after filtering. Right: The microtile decomposition is computed exactly per key point (geometric corners) and then transferred to the voxelized representation 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.*

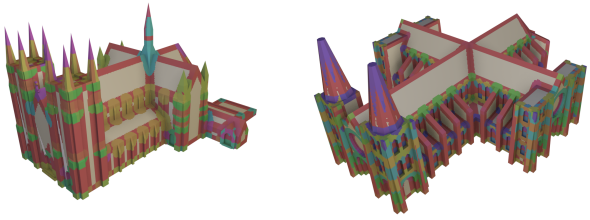

Figure 8: *Microtile decompositions of two test models downloaded from thingiverse.com. The cathedral on the left was decomposed in 38s into 931 microtiles of 527 classes. The church on the right was decomposed in 3min into 2276 microtiles of 395 classes.*

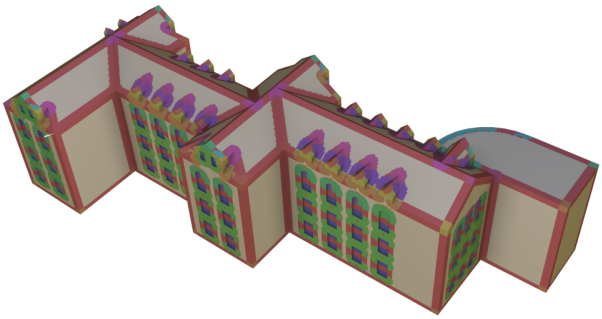

Figure 9: *It took 28s to decompose the above building on into 921 microtiles of 196 classes.*

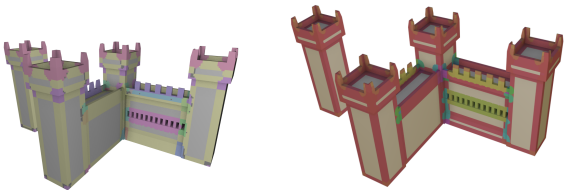

Figure 10: *Left: Output of the naïve decomposition algorithm for a castle with 4 towers. Run-time was approximately 1 hour. Right: the new algorithm is able to compute more detailed decomposition in 52s.*

pairs of spherical r-neighborhoods, and these operations are easier to distribute across multiple 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.

4.6 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 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 4.3). 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 to efficiently decompose non-trivial input models. The resulting decomposition does not suffer from artifacts at tile boundaries and the better efficiency allows computing a more-detailed decomposition. Note, that the visualization artifacts in the figures (e.g. Figure 8 and Figure 9) are caused by the use of a low resolution global 3D texture to color the individual building blocks.

We evaluate the new decomposition algorithm on mostly clean input models. The approximate neighborhood matching around individual features allows to handle slight deformations and typical inconsistencies in the geometric representation caused by the floating point representation of symmetric vertices and transformation matrices. However, we treat discrepancies above a user-defined threshold in the pairwise distances of r-neighborhoods as substantial and

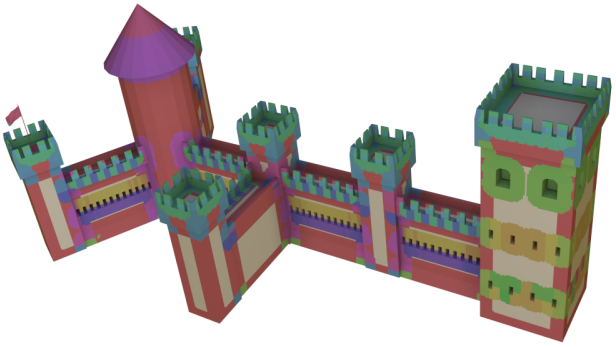

Figure 11: *This castle model was decomposed in 4min into 2078 microtiles of 962 classes. We tested 2 million r-symmetry candidates – triplets of two features and a transformation.*

did not identify the corresponding features as equivalent.

4.7 Discussion

In this document we showed how to lift important limitation of the microtile analysis introduced in [Kalojanov et al. 2012]. The main theoretical insight here is that although r-symmetry and r-similarity are defined per (infinitely-small) 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 procedural modeling. While there certainly are other methods for symmetry detection and different definitions of building blocks, so far only the microtiles, introduced here, allow to systematically compute and characterize 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.

References

BOKELOH, M., BERNER, A., WAND, M., SEIDEL, H. P., AND SCHILLING, A. 2009. Symmetry detection using feature lines. *Computer Graphics Forum 28*, 2, 697–706.

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

BOKELOH, M., WAND, M., KOLTUN, V., AND SEIDEL, H.-P. 2011. Pattern-aware shape deformation using sliding dockers. *ACM Trans. Graph. 30* (Dec.), 123:1–123:10.

GELFAND, N., AND GUIBAS, L. 2004. Shape segmentation using local slippage analysis. In *Proc. Symp. Geom. Processing*.

KALOJANOV, J., BOKELOH, M., WAND, M., GUIBAS, L., SEIDEL, H.-P., AND SLUSALLEK, P. 2012. Microtiles: Extracting building blocks from correspondences. *Computer Graphics Forum 31*, 5, 1597–1606.

LIPMAN, Y., CHEN, X., DAUBECHIES, I., AND FUNKHOUSER, T. 2010. Symmetry factored embedding and distance. *ACM Transactions on Graphics (SIGGRAPH 2010)* (July).

MITRA, N. J., GUIBAS, L. J., AND PAULY, M. 2006. Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph. 25*, 3, 560–568.

MITRA, N. J., PAULY, M., WAND, M., AND CEYLAN, D. 2012. Symmetry in 3d geometry: Extraction and applications. In *EUROGRAPHICS State-of-the-art Report*.