A Connection between Partial Symmetry and Inverse Procedural Modeling

Martin Bokeloh* Max-Planck-Institut Informatik
Michael Wand* Saarland University and Max-Planck-Institut Informatik
Hans-Peter Seidel* Max-Planck-Institut Informatik

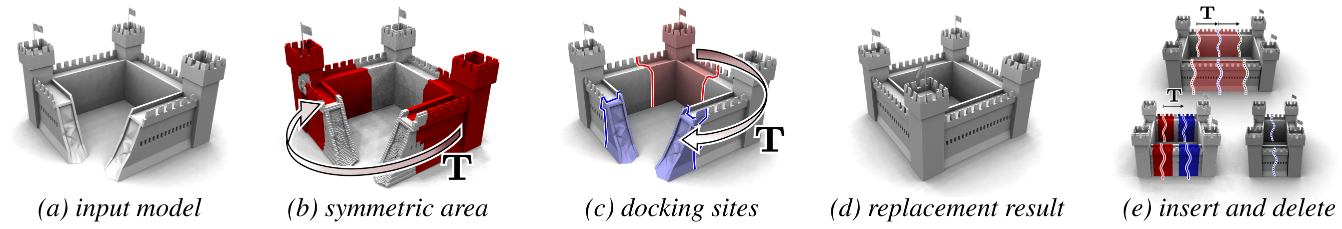


Figure 1: *(a) We compute shape modification operations by examining the partial symmetry structure of a 3D model. (b) Symmetric regions are marked in red. (c) A set of symmetric curves that cuts the model into two pieces yields a docking site that corresponds to (d) a replacement operation. (e) A similar construction yields insertions and deletions. Lower left: input with symmetry (red: source, blue: target).*

Abstract

In this paper, we address the problem of *inverse* procedural modeling: Given a piece of exemplar 3D geometry, we would like to find a set of rules that describe objects that are similar to the exemplar. We consider local similarity, i.e., each local neighborhood of the newly created object must match some local neighborhood of the exemplar. We show that we can find explicit shape modification rules that guarantee strict local similarity by looking at the structure of the partial symmetries of the object. By cutting the object into pieces along curves within symmetric areas, we can build shape operations that maintain local similarity by construction. We systematically collect such editing operations and analyze their dependency to build a shape grammar. We discuss how to extract general rewriting systems, context free hierarchical rules, and grid-based rules. All of this information is derived directly from the model, without user interaction. The extracted rules are then used to implement tools for semi-automatic shape modeling by example, which are demonstrated on a number of different example data sets. Overall, our paper provides a concise theoretical and practical framework for inverse procedural modeling of 3D objects.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling— [I.2.10]: Artificial Intelligence—Vision and Scene Understanding

Keywords: inverse procedural modeling, geometry synthesis, modeling by example

*e-mail: {mbokeloh,mwand,hpseidel}@mpi-inf.mpg.de

1 Introduction

Nowadays, model creation is one of the main bottlenecks in practical applications of 3D computer graphics. Creating high quality 3D models is a time consuming task that requires substantial artistic and technical skills. Consequently, a lot of recent work has focused on "intelligent" modeling tools that to some extend "understand" the structure of 3D shapes [Bhat et al. 2004; Sharf et al. 2004; Lagae et al. 2005; Zhou et al. 2006; Merrell 2007; Kraevoy et al. 2008; Gal et al. 2009].

In this paper, we propose a novel approach to the semi- and fully automatic creation of 3D models that are similar to a piece of example geometry. Unlike most previous work, our approach constructs a set of explicit, procedural rules that encode how to build such objects efficiently, leading to an *inverse procedural modeling* system [Aliaga et al. 2007; Šťava et al. 2010]. We formally guarantee that each newly created object is *r-similar* to the input exemplar $\mathcal{S}$: Analogous to non-parametric texture synthesis [Efros and Leung 1999], we demand that any point on the new object matches some point on the exemplar within a local neighborhood of radius r. The key observation of our paper is that the partial symmetries of an object reveal a set of *shape operations* that alter the object while guaranteeing strict r-similarity: Assume we fix a transformation $\mathbf{T}$ and look at the regions of an object (Figure 1a) that are symmetric under this transformation (Figure 1b). We might find non-symmetric regions that are separated from the rest of the model by some symmetric area. We call these regions *dockers* and a curve through a symmetric region that cuts out a docker a *docking site* (Figure 1c). As the docking site geometry matches, we can exchange the corresponding dockers while maintaining similarity to the input exemplar. In general, we obtain operations that insert, delete, or replace pieces of $\mathcal{S}$ (Figure 1d,e). We examine the dependency of such operations and encode the result in a *shape grammar* that encodes a set of r-similar objects. We consider three variants: General rewriting systems, context-free grammars, and supplemental grid structured production rules. We implement the described analysis framework in a numerically robust way, handling general triangle meshes as well as point cloud data from 3D scanners, and demonstrate prototypical tools for shape modeling by example.

To the best of our knowledge, our technique is the first that is able to compute shape grammars for general 3D surfaces from example geometry without any user interaction (the similarity radius r is the only parameter). We provide a theoretical framework that establishes an interesting link between partial symmetry of an object $\mathcal{S}$ and a space of objects similar to $\mathcal{S}$. It provides strict formal guarantees: All computed models are strictly r-similar to the exemplar. In particular, any topologically consistent, closed input manifold will yield output models with these properties. However, in order to provide such strict guarantees, our formal framework requires models that have perfect partial symmetries, which is currently still a main limitation of the presented approach.

2 Related Work

Texture and geometry synthesis: Our approach is motivated by non-parametric texture synthesis [Efros and Leung 1999; Hertzmann et al. 2001; Kwatra et al. 2003], which optimizes for similarity of local, overlapping neighborhoods to corresponding regions in the exemplar image. Being formulated as an optimization problem, this leads to a hard Markov random field (MRF) inference problem. Texture synthesis has also been applied to 2D vector graphics [Barla et al. 2006; Ijiri et al. 2008] and the notion of local r-similarity of neighborhoods has been generalized to 3D geometry by [Rustamov 2008]. Texture synthesis can be applied to synthesize 3D geometry by discretizing a base surface and synthesizing details on top of it [Lai et al. 2005; Nguyen et al. 2005; Chen and Meng 2009; Zhou et al. 2006; Zelinka and Garland 2006]. This requires that the coarse scale base geometry is given as user input. Alternatively, one can discretize the ambient space itself, synthesizing occupancy in space. Techniques include voxel models [Bhat et al. 2004] and implicit functions [Sharf et al. 2004; Lagae et al. 2005]. However, it is difficult to find good solutions based on heuristic MRF optimization; creating closed and well defined geometry is more challenging than synthesizing plausible 2D images. None of the known methods have so far demonstrated results where large scale models with complex structure, such as complete buildings, are synthesized from scratch. [Merrell 2007] propose a related algorithm that expects building blocks aligned with a regular grid as input. These are then placed automatically with consistency across grid faces, again involving a discrete MRF labeling problem. The technique can handle arbitrary boundary conditions but, unlike our approach, building blocks are required as input rather than output and the regular grid structures limits the design space significantly. This has been addressed in [Merrell and Manocha 2008], where cells are formed by intersecting planes through faces of the exemplar model, implicitly creating the grid structure. However, the approach is limited to very low complexity input exemplars (examples in the paper have up to 39 input faces). [Cabral et al. 2009] examine a related idea: User specified building blocks and a connectivity graph are the input and the system then optimizes vertex positions and textures to form a closed model. It requires the building blocks and their interconnection rules as input, while our paper aims at computing this type of information automatically.

Shape grammars and inverse procedural modeling: Grammar-based modeling is one of the most successful procedural modeling paradigms. Applications include plant modeling [Prusinkiewicz and Lindenmayer 1990] and modeling of cities [Parish and Müller 2001] and buildings [Wonka et al. 2003; Müller et al. 2006]. *Inverse procedural modeling* is referring to the reverse process, where rules (such as shape grammars) have to be derived from example geometry. This goes beyond geometry synthesis in the sense that it not just creates similar models from exemplars but also describes the structure of the space of such models. An early approach is the work of [Hart and Flynn 1997] who derive fractal branching rules for L-systems from 2D example graphics by geometric hashing, however, being limited to simple L-systems with a few rules only. In more recent work, regular patterns have been computed from example geometry [Pauly et al. 2008]. In [Mitra and Pauly 2008], this technique is used for creating variants from example models by changing the replication frequency or editing symmetric pieces simultaneously. Our approach is based on a different idea: We do not replicate symmetric parts but look for symmetric area (docking sites) enclosing non-symmetric geometry (dockers) such that the non-symmetric parts can be replaced. Grid structures are found if dockers recursively contain matching docking sites. This analysis works with and without grid structures present in the example; it can even create grids of dockers from only a single example docker, if appropriate recursive docking sites are found. [Yeh and Měch 2009] analyze 2D vector graphics to detect complex 1D patterns along curves with secondary structure. Very recently, [Št'ava et al. 2010] extend this idea to detect hierarchies of patterns, yielding an L-system describing the example geometry, similar to the context free representation provided by our algorithm. Our scenario is more complicated as it deals with 3D surfaces that need to be assembled in a consistent way, without holes in the surface. In contrast, 2D vector graphics can just be composited arbitrarily. A lot of work has been presented that uses procedural rules to fit geometry to image input rather than geometry [Aliaga et al. 2007; Müller et al. 2007; Neubert et al. 2007; Tan et al. 2007; Xiao et al. 2009], which is an even harder inverse problem. These techniques use predefined classes of rules (such as hierarchical regular subdivisions of facades [Müller et al. 2007]) or a significant amount of user input to facilitate image interpretation but do not attempt to create shape grammars automatically from scratch.

Smart shape deformation: Recently, a number of shape deformation techniques have been proposed that try to preserve important object features, including non-local properties such as symmetry or orientational alignment [Kraevoy et al. 2008; Gal et al. 2009]. Deformation methods are fully orthogonal to our approach: Unlike our technique, they cannot insert or delete elements, nor change the topological arrangement of building blocks the model is composed of. However, our technique currently does not perform any additional deformation. A related approach is modeling by example [Funkhouser et al. 2004; Sharf et al. 2006; Kraevoy et al. 2007]: the system semantically identifies similar pieces that are composited according to user input into a joint model by deformable matching and cutting. This allows for more general changes but still needs user guidance to assemble the pieces.

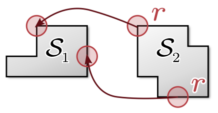
3 Formal Model

In this section, we present our theoretical framework, proceeding in the following steps: First, we define the basic notions of *symmetry* and *similarity* (Subsection 3.1). Then, we describe how to compute *docking sites*, *dockers*, and how to construct the associated *shape operations* (Subsection 3.2). Afterwards, we bring these elements into a canonical form in Subsection 3.3. Finally, we discuss how to combine the obtained rules into a *shape grammar* (Subsection 3.4).

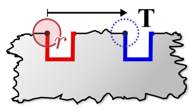
3.1 Similarity and Symmetry

Input and neighborhoods: We assume that we are given an *input exemplar* $\mathcal{S} \subset \mathbb{R}^3$. For simplicity, we assume a finite, piecewise smooth surface. For every $\mathbf{x} \in \mathcal{S}$, we define the *r-neighborhood* $N_r^{\mathcal{S}}(\mathbf{x}) = N_r(\mathbf{x}) := \{\mathbf{y} \in \mathcal{S} \mid dist(\mathbf{x}, \mathbf{y}) \leq r\}$, where $dist$ is a metric on $\mathcal{S}$. In the following, we use the intrinsic distance within $\mathcal{S}$ to correctly handle close pieces that are topologically disconnected.

Transformations: In the following, symmetry and similarity are defined with respect to a *group of admissible transformations* $\mathcal{T}^+$ consisting of continuous, injective functions $\mathbf{T} : \mathcal{S} \to \mathbb{R}^3$. We denote the *transformation of a set* $\mathcal{A}$ as $\mathbf{T}(\mathcal{A}) := \{\mathbf{T}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{A}\}$. For the rest of this paper, we restrict ourselves to the group of rigid



(a) r-similarity



(b) r-symmetry

Figure 2: *(a) r-similarity: every point of $\mathcal{S}_2$ is locally similar to a point of $\mathcal{S}_1$, within a radius of r. (b) r-symmetry is defined analogously: under a fixed transformation $\mathbf{T}$, symmetric points must be similar within a radius of r.*

motions. However, most of our framework could be easily generalized to different notions of symmetry and similarity, induced for example by invertible affine mappings or by intrinsic isometries.

We can now define our notion of local similarity of objects: An object is similar to an exemplar if each local neighborhood of the object can be found somewhere in the exemplar (Figure 2a). We formalize this concept in the notion of r-similarity:

r-similarity. Given two surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$, $\mathcal{S}_2$ is *r-similar* to $\mathcal{S}_1$ if and only if for all $\mathbf{y} \in \mathcal{S}_2$ there exist a point $\mathbf{x} \in \mathcal{S}_1$ and a mapping $\mathbf{T} \in \mathcal{T}^+$ such that:

(1) $\mathbf{T}(\mathbf{x}) = \mathbf{y}$ and $\mathbf{T}(N_r^{\mathcal{S}_1}(\mathbf{x})) \subseteq \mathcal{S}_2$.
(2) For all $\tilde{\mathbf{x}} \in N_r^{\mathcal{S}_1}(\mathbf{x}) : \mathbf{x} \cong \mathbf{T}(\tilde{\mathbf{x}})$,

where "$\cong$" means that the local topology is preserved with respect to the surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$ where the points are contained in (formally: there exist infinitesimal neighborhoods that are homeomorphic). Condition (1) ensures that we find a geometrically matching neighborhood in $\mathcal{S}_1$ for every point of $\mathcal{S}_2$. Condition (2) makes sure that the topology matches as well. In particular, we cannot map a manifold interior point to a boundary point, or an interior point to a T-junction, and vice versa.

Please note that the definition of r-similarity is not symmetric. We always place the r-neighborhood on the exemplar surface $\mathcal{S}_1$, never on the surface that $\mathbf{T}$ maps to. This is important for generalizations to non-rigid mappings where $\mathbf{T}$ might contains scaling that alters the size of the neighborhood. Analogous to r-similarity, we define the notion of r-symmetry, where we map neighborhoods within a single surface. We start by defining infinitesimal symmetry:

Symmetry under a transformation T: We denote the set of all points of $\mathcal{S}$ that are symmetric w.r.t. a transformation $\mathbf{T}$ as:

$$\xi(\mathbf{T}) := \{\mathbf{x} \in \mathcal{S} \mid \mathbf{T}(\mathbf{x}) \in \mathcal{S} \text{ and } \mathbf{x} \cong \mathbf{T}(\mathbf{x})\},$$

Two points of $\mathcal{S}$ are symmetric under a transformation $\mathbf{T}$ if $\mathbf{T}$ maps between them and their infinitesimal local neighborhood is topologically equivalent. Accordingly, the non-symmetric points w.r.t. $\mathbf{T}$ are denoted by $\bar{\xi}(\mathbf{T}) := \mathcal{S} \setminus \xi(\mathbf{T})$.

r-symmetry under a transformation T: The set of all points that are *r-symmetric* under a transformation $\mathbf{T}$ is given by:

$$\xi_r(\mathbf{T}) := \{\mathbf{x} \in \mathcal{S} \mid \text{all } \tilde{\mathbf{x}} \in N_r(\mathbf{x}) \text{ are symmetric under } \mathbf{T}\}$$

This means, a point is r-symmetric iff its whole r-neighborhood is symmetric under the same transformation (Figure 2b). In other words, we obtain the r-symmetric points by an erosion operation of radius r on the set of points that map from $\mathcal{S}$ back to $\mathcal{S}$ under $\mathbf{T}$. We will use this observation in Section 4 to devise a simple and efficient algorithm for computing the r-similarity structure of an object. The algorithm will actually work on a discrete set of candidate transformations $\mathbf{T}$, constructed by matching surface features, for which it computes the symmetric area. For now, we will just assume that we know $\xi_r(\mathbf{T})$ for all transformations $\mathbf{T}$ that might be of interest and discuss the details of symmetry detection later.

3.2 Dockers, Docking Sites and Shape Operations

Our goal is now to determine shape operations that modify an exemplar shape while maintaining r-similarity to the original. The key insight of this paper is that the symmetry structure of an object reveals a rich set of such operations as well as their interdependence. For now, we will consider one fixed transformation $\mathbf{T}$ and examine the r-symmetry structure of $\mathcal{S}$ with respect to $\mathbf{T}$. How to combine the result from different transformations within a shape grammar will be discussed later in Subsection 3.4.

$\xi_r(\mathbf{T})$ can be regarded as a binary function on the exemplar $\mathcal{S}$ that marks symmetric regions on the surface (Figure 1b shows an example visualizing $\xi_r(\mathbf{T})$ as red area). Obviously, any subset $\mathcal{X} \subseteq \xi_r(\mathbf{T})$ of a symmetric region can be exchanged with its corresponding geometry $\mathbf{T}(\mathcal{X})$ without changing the geometry at all. However, if we find a piece of non-symmetric geometry that is divided from the rest of the model by symmetric area, we can use the symmetric region as a *docking site* to replace some geometry with different geometry while maintaining r-similarity to the exemplar. We formalize this observation in the following definition:

Docking sites and dockers: A set of surface curves $\boldsymbol{\alpha} \subseteq \mathcal{S}$ is called a *docking site* with respect to a transformation $\mathbf{T}$, if the following three properties hold (see Figure 3a,b):

- $\boldsymbol{\alpha}$ is r-symmetric under $\mathbf{T}$: $\boldsymbol{\alpha} \subseteq \xi_r(\mathbf{T})$.
- $\boldsymbol{\alpha}$ partitions the model into two topologically disconnected pieces $\mathcal{D}_{\boldsymbol{\alpha}}$ and $\mathcal{R}_{\boldsymbol{\alpha}}$. This means, any continuous path in $\mathcal{S}$ between the two pieces must intersect $\boldsymbol{\alpha}$. The piece $\mathcal{D}_{\boldsymbol{\alpha}}$ contains geometry that is not symmetric under $\mathbf{T}$: $\mathcal{D}_{\boldsymbol{\alpha}} \not\subseteq \xi_r(\mathbf{T})$. We call $\mathcal{D}_{\boldsymbol{\alpha}}$ a *docker* for *docking site* $\boldsymbol{\alpha}$
- $\mathbf{T}(\boldsymbol{\alpha})$ also partitions the model into two topologically disconnected pieces $\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$ and $\mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})}$. We call $\mathbf{T}(\boldsymbol{\alpha})$ the *secondary docking site* and $\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$ the *secondary docker* of $\boldsymbol{\alpha}$.

For clarity we will sometimes refer to the original, non-transformed docking sites (and dockers) as *primary* docking sites (and *primary* dockers), as opposed to secondary docking sites. By convention, the docking site $\boldsymbol{\alpha}$, the docker $\mathcal{D}_{\boldsymbol{\alpha}}$ and the remaining geometry $\mathcal{R}_{\boldsymbol{\alpha}}$ are defined to be disjoint. The same applies to the secondaries.

Figure 3b shows an example of a partitioning of the model into the pieces $\mathcal{D}_{\boldsymbol{\alpha}}$, $\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$, $\mathcal{R}_{\boldsymbol{\alpha}}$, and $\mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})}$. Obviously, this is only possible if $\boldsymbol{\alpha}$ and $\mathbf{T}(\boldsymbol{\alpha})$ cut the model into disconnected pieces (Figure 3a). Figure 3c,d show how we can use docking sites and dockers to modify geometry: The docking site itself is situated within symmetric geometry. Therefore, we can cut the model through the docking sites and replace the docker with the secondary docker, or vice versa. By construction, the boundary is r-similar so that the pieces fit together. Formally, every shape operation is performed by the same rule, a *replacement* of dockers:

Replacement shape operations: Let $\boldsymbol{\alpha}$ be a docking site with respect to transformation $\mathbf{T}$. A *shape operation*, $\mathtt{op}_{\boldsymbol{\alpha},\mathbf{T}}$ is given by

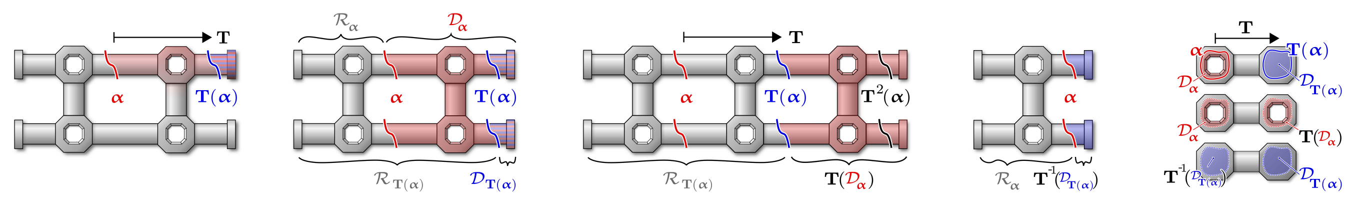
$$\mathtt{op}_{\boldsymbol{\alpha},\mathbf{T}} : \mathcal{S} \rightarrow \mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})} \cup \mathbf{T}(\mathcal{D}_{\boldsymbol{\alpha}}) \cup \mathbf{T}(\boldsymbol{\alpha}).$$

It replaces the secondary docker with the transformed primary docker (Figure 3c and Figure 3e, middle row). Correspondingly, there is also a *secondary shape operation* $\mathtt{op}_{\mathbf{T}(\boldsymbol{\alpha}),\mathbf{T}^{-1}}$

$$\mathtt{op}_{\mathbf{T}(\boldsymbol{\alpha}),\mathbf{T}^{-1}} : \mathcal{S} \rightarrow \mathcal{R}_{\boldsymbol{\alpha}} \cup \mathbf{T}^{-1}(\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}) \cup \boldsymbol{\alpha},$$

which replaces the secondary docker with the primary docker, transformed accordingly (see Figure 3d and Figure 3e, lower row).

Obviously, a secondary operation for a docking site $\boldsymbol{\alpha}$ and transformation $\mathbf{T}$ is identical to the primary replacement operation for the docking site $\mathbf{T}(\boldsymbol{\alpha})$ and transformation $\mathbf{T}^{-1}$. Please note that our


T
α
T(α)
$\mathcal{R}_\alpha$
$\mathcal{D}_\alpha$
$\mathcal{R}_{T(\alpha)}$
$\mathcal{D}_{T(\alpha)}$
$T^2(\alpha)$
$T(\mathcal{D}_\alpha)$
$T^{-1}(\mathcal{D}_{T(\alpha)})$

(a) no docking site *(b) proper docking site* *(c) insert* *(d) delete* *(e) replace*

Figure 3: *Definition of a docking site. (a) The first part of the definition is met, but the docking site does not divide the model into two disconnected pieces. Therefore, no valid operation results. (b) A valid docking site, meeting all three criteria. (c),(d) Dockers can be exchanged while keeping the model r-similar to $\mathcal{S}$. If primary and secondary dockers are contained in each other, this results in inserting or deleting symmetric pieces. (e) Otherwise, geometry is just replaced; please note that this example shows a different geometry.*

notation for the shape operations already makes use of this observation. The consequence in practice is that we can restrict ourselves to collecting only primary operations; we will obtain the secondary operations automatically by considering the corresponding symmetries under the inverse transformation.

Classification: We can distinguish further between different effects of the primary operations by looking at the topological relation of primary and secondary dockers. We obtain three cases that lead to different types of operations, as illustrated in Figure 3c-e:

- **Insert:** $\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})} \subset \mathcal{D}_{\boldsymbol{\alpha}}$. The secondary docker is a subset of the primary docker (Figure 3c).
- **Delete:** $\mathcal{D}_{\boldsymbol{\alpha}} \subset \mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$. The primary docker is a subset of the secondary docker (Figure 3d).
- **Replace:** All other cases (Figure 3e).

Replace operations just substitute parts of the geometry with an alternative piece of geometry. Insert and delete operations are special cases of replacements that replace a piece of geometry with two or zero copies of itself so that they effectively grow or shrink the object. These operations (and combinations of them) are useful for "smart resizing" of 3D objects; we will make use of this later in the applications section. Please note that the replace shape operation is still possible if the dockers overlap partially, i.e. $\mathcal{D}_{\boldsymbol{\alpha}} \cap \mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})} \neq \emptyset$. However, the operations become dependent (which also frequently occurs for operations derived from two different transformations). Resolving dependencies is discussed in Subsection 3.4.

Collision avoidance: An important aspect we neglect so far is that of *collisions*. We have to make sure that $\mathbf{T}(\mathcal{D}_{\boldsymbol{\alpha}})$ and $\mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})}$ do not *collide*, i.e., have a non-zero distance everywhere, except from the docking site where they meet. If this is not the case but the a newly inserted piece collides with already existing geometry, the operation might not be valid and we do not perform the shape operation. We can now formulate the main result of this subsection:

Maintaining r-similarity: Given an input surface $\mathcal{S} \subset \mathbb{R}^3$ and an r-docking site $\boldsymbol{\alpha}$ under transformation $\mathbf{T}$. If the shape operation $\mathtt{op}_{\boldsymbol{\alpha},\mathbf{T}}$ is not colliding, it will create a result that is r-similar to $\mathcal{S}$.

3.3 Elementary Docking Sites

So far, we have shown how to extract shape operations from the symmetry structure of an exemplar surface. However, there are a few technical problems left: First, we can get an infinite number of equivalent operations by moving the docking site within the symmetric region (Figure 4a). We thus need to normalize the construction so that we get a canonical representation for a shape operation. Even after doing this, the number of shape operations for a single transformation $\mathbf{T}$ might still be exponential in the number of non-symmetric "docker" regions, because the docking site can select an arbitrary subset of such regions that are located within a common symmetric region (Figure 4b). This obviously prevents us from compactly encoding the set of all discovered shape operations. *Elementary docking sites* (Figure 4c) address both problems:

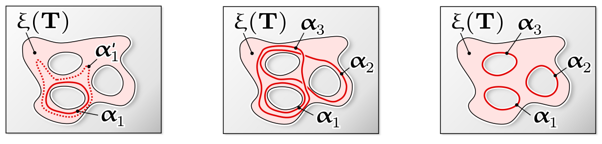
Elementary docking sites: An *elementary docking site* is a docking site for which the closure of its docker is minimal with respect to set inclusion. Shape operations derived from elementary docking sites are called *elementary shape operations*.

Intuitively, we obtain an elementary docking site by shrinking the docker enclosed by the docking site as much as possible, enclosing only the non-symmetric region, at the boundary between r-symmetric and non-r-symmetric area (with respect to a certain transformation $\mathbf{T}$). In addition, we are not allowed to enclose more non-r-symmetric regions than necessary to divide the model into two pieces, because otherwise, the docker would not be minimal. We now discuss how to compute elementary docking sites. For this, and all of the following, we will always assume that we have a finite number of non-symmetric regions, i.e., $\overline{\xi_r(\mathbf{T})}$ consists of a finite number of connected sets, each with continuous boundary curves. For practical models such as triangle meshes, this assumption is always met. The idea for the algorithm is rather simple: For any valid docker, we need to make sure that a docker is separated completely from the remaining geometry by the docking sites, which can be solved by simple region growing. However, we need to simultaneously assure that the same is true for the corresponding secondary dockers, which therefore requires alternated region growing in the primary and secondary domain.

Algorithm: Computing elementary docking sites

Input: We are given an input model $\mathcal{S}$, a transformation $\mathbf{T}$, and the symmetry structure $\xi_r(\mathbf{T})$.

Algorithm: We start by computing the boundary curves $\partial\overline{\xi_r(\mathbf{T})}$ of the non-symmetric regions using region growing, as well as the secondary boundaries $\mathbf{T}(\partial\overline{\xi_r(\mathbf{T})})$, which must exist due to symmetry. This partitions the model into a set of primary and secondary non-symmetric regions, which will be combined to valid and elementary primary and secondary dockers next. We build a graph that encodes the dependencies of these regions: We connect primary and secondary non-symmetric regions if they share a boundary curve where the primary boundary maps to the secondary under $\mathbf{T}$. The important observation is that this does not need to be symmetric. For example, the secondary can map to two boundaries while the primaries only map to one each, as shown in Figure 5. In order to obtain a valid combination we compute the connected components in this dependency graph. All primary regions within a connected component form one primary docker, and all secondary regions the secondary docker. Their boundaries to the symmetric area are the docking sites. These pieces are elementary because no smaller set can cut the model into two pieces on both primary and secondary side. Figure 5 shows how we subsequently add dependent pieces to extract a valid pair of primary/secondary dockers.



(a) equivalent sites *(b) combinatorial explosion* *(c) elementary sites*

Figure 4: *General vs. elementary docking sites: (a) The docking sites α_1 and α'_1 are equivalent. (b) By combining multiple non-symmetric regions, we can obtain an exponential number of non-equivalent docking sites. (c) Elementary docking sites form a canonical set for composing more complex operations.*

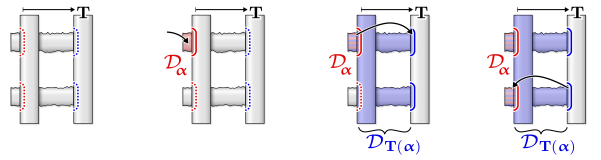
One can easily show that elementary dockers for the same transformation $\mathbf{T}$ are always disjoint. Because of this, it is easy to see that any general shape operation with respect to a fixed transformation $\mathbf{T}$ can always be achieved by a combination of elementary shape operations. This means, for n different non-symmetric regions, we only need to store $\mathcal{O}(n)$ elementary shape operations, rather than in the worst case $2^n - 1$ non-elementary ones.

3.4 Extracting Shape Grammars

So far, we have found a number of operations that allow us to change the input shape $\mathcal{S}$ while keeping it r-similar to the original. However, we can only change the original input once: the changes might alter the situation such that the original operations are not applicable anymore. In order to enable the execution of multiple, consecutive shape operations, we therefore need to to determine their interdependency, and if necessary adapt the executed operation accordingly. In the following, we will describe three different models that build sets of rules that combine multiple shape operations. We start with a straightforward construction that yields a general Chomsky type-0 grammar (general rewriting system) based on shape matching. This model is the most expressive, but its structure is the least computationally accessible. Therefore, in the next step, we compute a context-free subset of this grammar, which is easier to handle in applications. Lastly, we add non-context free grid-based replication rules with multiple degrees of freedom, which are very useful for analyzing real-world objects [Müller et al. 2007].

3.4.1 A Basic Shape Matching Grammar

Assume that we are given a set of shape operations for an exemplar $\mathcal{S}$ and we want to execute multiple of these operations subsequently while maintaining r-similarity to $\mathcal{S}$. The important observation at this point is that we can apply a shape operation in any place where we find a suitable docking site. "Suitable" means that the new docking site is related to the original one by an admissible transformation $\mathbf{T} \in \mathcal{T}^+$. In that case, we can just transform the docker of the shape operation by $\mathbf{T}$ to match the new docking site. As the admissible transformations form a group by definition, the compound transformation will automatically be admissible as well and the transformed operation still guarantees r-similarity. This directly yields a shape matching grammar: Before each operation, we have to search the model for matching docking sites (using rigid shape matching, in our case) and then obtain a selection of applicable operations. Obviously, this approach is costly to compute and incompatible with standard procedural modeling tools. In addition, the structure of the language might be very complex. As a general rewriting system (Chomsky-0 grammar), most properties of the language are uncomputable and thus inaccessible for controlling the modeling process. Therefore, we have to extract a more manageable subset. This is the subject of the next subsection:



$\mathbf{T}$
$\mathbf{T}$
$\mathcal{D}_{\boldsymbol{\alpha}}$
$\mathbf{T}$
$\mathcal{D}_{\boldsymbol{\alpha}}$
$\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$
$\mathbf{T}$
$\mathcal{D}_{\boldsymbol{\alpha}}$
$\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$

(a) symm. borders *(b) no docker yet* *(c) dep. secondary* *(d) final output*

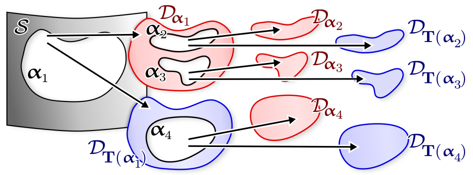
Figure 5: *Computing elementary docking sites: (a) Transformation $\mathbf{T}$ and corresponding boundaries of r-symmetry. (b) The red piece is non-symmetric. (c) It is not a docker (yet) because its secondary boundary does not split the model. (d) Adding the missing dependent piece yields valid primary/secondary dockers.*

3.4.2 Context Free Grammars

In a context free grammar, we do not perform shape matching but use *non-terminal symbols* to identify a space where further geometry can be plugged in. The geometry itself is encoded as *terminal symbols*. A set of *production rules* describes which terminal and non-terminal pieces can be plugged into the space designated by each non-terminal piece, leading to a hierarchical structure of insertions of pieces. In our framework, *docking sites* represent non-terminal symbols because alternative geometry can be plugged into the regions they enclose (Figure 6). *Dockers* are the pieces being plugged in, which in turn might contain further sub-docking-sites. Therefore, the geometry of a docker, minus that enclosed by sub-docking-sites, corresponds to terminal symbols, and the contained docking sites are hierarchically dependent non-terminals. We will reflect this fact in our notation and use non-terminals and docking sites interchangeably, as well as dockers and terminal symbols.

Using context-free production rules means that we need to be able to tell a priori whether a certain piece of geometry that is tagged with a non-terminal node will allow for the insertion of certain alternative pieces or not. We cannot retract from this decision later on because we do not perform any online shape matching. Any conflicts between rules that try to alter the same piece of geometry must be already resolved during the construction of the formal language. Obviously, potential conflicts are only created by designating docking sites. Identifying a docker and storing its terminal geometry cannot create conflicts, no matter how these dockers overlap on the exemplar. However, if we put a docking site on a piece of geometry, we must make sure that the docker area it encloses does not intersect with further docking sites, because the geometry within these bounds is subject to change. More precisely, if we create the production rule for a single docker, the docking sites (i.e., the non-terminals) within this rule must not overlap. We use the following formal classification: Two docking sites $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ are *conflicting* iff $\boldsymbol{\alpha}_1 \cap \boldsymbol{\alpha}_2 \neq \emptyset$, i.e., if the *docking sites* intersect. $\boldsymbol{\alpha}_2$ is *hierarchically dependent* on $\boldsymbol{\alpha}_1$ iff $\mathcal{D}_{\boldsymbol{\alpha}_2} \subset \mathcal{D}_{\boldsymbol{\alpha}_1}$, i.e., the *dockers* are contained in each other. Obviously, hierarchically dependent docking sites are not conflicting in the sense of this definition, but we still cannot combine the rules arbitrarily but need to make sure that the docker they are based on does already exist. This idea leads to a hierarchical construction algorithm. The main idea is to first identify hierarchical dependence of docking sites. Then we iteratively consider each docker and handle all docking sites it directly contains. If these docking sites are conflicting, we remove some of them until all conflicts are resolved. As there are multiple alternative ways to achieve this, we in general obtain several production rules for this docker. We now look at this more in detail:

Constructing a context-free shape grammar: We first build a tree that encodes the hierarchical dependence of the docking sites. A node $\boldsymbol{\alpha}$ is contained in the subtree of a node $\boldsymbol{\beta}$ if and only if $\boldsymbol{\alpha}$ is completely contained in $\mathcal{D}_{\boldsymbol{\beta}}$. This always yields a unique tree



$\mathcal{S}$
$\boldsymbol{\alpha}_1$
$\mathcal{D}_{\boldsymbol{\alpha}_1}$
$\boldsymbol{\alpha}_2$
$\boldsymbol{\alpha}_3$
$\boldsymbol{\alpha}_4$
$\mathcal{D}_{\boldsymbol{\alpha}_2}$
$\mathcal{D}_{\boldsymbol{\alpha}_3}$
$\mathcal{D}_{\boldsymbol{\alpha}_4}$
$\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha}_2)}$
$\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha}_3)}$
$\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha}_4)}$
$\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha}_1)}$

Figure 6: *Extracting a hierarchy of docking sites and dockers yields a context free grammar.*

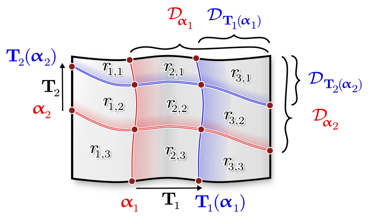
because the inclusion relation is tree structured by definition. After that, we look at each node in the tree from bottom up, starting at the leaf nodes. The docker of each leaf node yields a terminal node in our grammar that contains only fixed geometry and no further non-terminals. The docking site of each leave nodes is represented as a non-terminal node and we create the corresponding production rules that encode two alternatives each: inserting either the primary or the secondary docker. Now we go up one level in the hierarchy and build more complex rules: Let $\boldsymbol{\alpha}$ be an inner node, and $\mathcal{C}(\boldsymbol{\alpha})$ be the set of all docking sites that are direct children of this node. These children form areas where we can insert hierarchically dependent pieces (which in turn might contain further docking sites, i.e., further non-terminals). However, the docking sites in $\mathcal{C}(\boldsymbol{\alpha})$ will in general overlap, creating conflicts. Therefore, we create multiple alternative rules that each resolve the conflicts in a different way: We consider the graph of overlapping docking sites, where the docking sites in $\mathcal{C}(\boldsymbol{\alpha})$ are the vertices and an edge is inserted if the two sites are conflicting. In this graph, we compute the set of all maximal independent sets. This yields the set of all sets of docking sites that do not overlap each other. In order not to miss options, we also need to unfold the hierarchy of overlapping nodes and include these in the independent set computation. Each of the obtained independent sets $\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_k\}$ is free of collisions and is therefore converted into one production rule for the non-terminal $\boldsymbol{\alpha}_0$:

$$\boldsymbol{\alpha} \rightarrow \underbrace{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_k}_{\text{non-terminals}}, \underbrace{(\mathcal{D}_{\boldsymbol{\alpha}} \setminus (\mathcal{D}_{\boldsymbol{\beta}_1} \ldots \mathcal{D}_{\boldsymbol{\beta}_k}))}_{\text{terminal remainder geometry}}$$

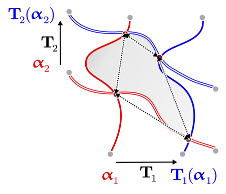
The production rule encodes that the non-terminal $\boldsymbol{\alpha}$ can be replaced by a set of non-conflicting docking sites and the remaining geometry not covered by these sites. For each independent set, one such separate rule is created. Once we have performed the operation for the primary docker of $\boldsymbol{\alpha}$, we repeat the same procedure for the secondary docker, which also fits into $\boldsymbol{\alpha}$.

Complexity problem: The strategy has a problem in practice: the number of maximal independent sets in a graph has an exponential worst-case lower bound of $\Omega(3^{n/3})$ sets for n nodes. Correspondingly, the algorithm might become impractical for complex inputs. Therefore, we use a bounded complexity approximation in practice: Instead of enumerating all independent sets, we sample the solution space randomly. We iteratively choose a random node and remove all colliding nodes until no more nodes are left. We make sure to start at a different node each time and limit the number of trials to a fixed constant (in our examples: 10). Thus the maximum number of computed production rules is fixed as well. In addition, we only unfold at most one hierarchy level; if such a node still collides, we dismiss all hierarchically contained docking sites as well. This approximation does not yield the largest possible context free sublanguage but it is very fast and produces good results in practice.

Improvements: In order to make the grammar more expressive, we perform shape matching between docking sites before constructing the grammar and identify all docking sites of the same shape. This means that not just the primary and secondary docker can be inserted into the corresponding non-terminal symbol but all dockers



$\mathcal{D}_{\boldsymbol{\alpha}_1}$
$\mathcal{D}_{\mathbf{T}_1(\boldsymbol{\alpha}_1)}$
$\mathbf{T}_2(\boldsymbol{\alpha}_2)$
$\mathbf{T}_2$
$\boldsymbol{\alpha}_2$
$r_{1,1}$
$r_{2,1}$
$r_{3,1}$
$r_{1,2}$
$r_{2,2}$
$r_{3,2}$
$r_{1,3}$
$r_{2,3}$
$r_{3,3}$
$\mathcal{D}_{\mathbf{T}_2(\boldsymbol{\alpha}_2)}$
$\mathcal{D}_{\boldsymbol{\alpha}_2}$
$\boldsymbol{\alpha}_1$
$\mathbf{T}_1$
$\mathbf{T}_1(\boldsymbol{\alpha}_1)$



$\mathbf{T}_2(\boldsymbol{\alpha}_2)$
$\mathbf{T}_2$
$\boldsymbol{\alpha}_2$
$\boldsymbol{\alpha}_1$
$\mathbf{T}_1$
$\mathbf{T}_1(\boldsymbol{\alpha}_1)$

(a) configuration for a grid patterns *(b) no valid grid*

Figure 7: *(a) Two colliding insert/delete operations divide the model into regions of nine different types. The corners are used once, the central piece is repeated on a 2-grid, and the rest is linearly repeated. (b) Counterexample: This arrangement of docking sites does not form a tileable grid cell.*

with similar docking site. In addition, we also try to avoid docking sites that are created by continuous symmetries, such as a window that can slide across a flat wall. We remove such rules by detecting slippable docking sites using the technique of [Gelfand and Guibas 2004]. We use this optional filter in all of our examples.

3.4.3 Regular Grids

Many real-world objects contain grid structures, such as a grid of $n_1 \times n_2$ windows in the facade of a building. In the following, we will call such structures k-grids, were k is the number of discrete degrees of freedom. A context free grammar cannot represent k-grids for $k \geq 2$ (unless we fix the repetition counts a priori, which is not useful in our application). Thus, we add a separate grid replication rule to our shape grammar that models this case.

1-grids: In our framework, 1-grids are identified by "insert" and "delete" shape operations: By definition, these operations are always dual to each other, and they mutually undo the effect of each other. In addition, the insert operation can be repeated an arbitrary number of times (up to collisions) because the inserted part by construction contains a docking site for another insertion.

2-grids: Grids with more than one degree of freedom show up as collisions of docking sites of multiple 1-grids. We first consider the case $k = 2$ for two insert type shape operations $\mathtt{op}_{\boldsymbol{\alpha}_1,\mathbf{T}_1}, \mathtt{op}_{\boldsymbol{\alpha}_2,\mathbf{T}_2}$. If the docking sites $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ intersect, the corresponding primary and secondary dockers intersect as well, due to symmetry. We classify the pieces by up to nine possible cases (Figure 7a). We choose an index of 1 for the remainder geometry, 2 for within the docking site but outside the secondary docker and 3 for inside the primary and the secondary docker. Accordingly, we label the pieces by $r_{1,1}$ up to $r_{3,3}$, taking both operations into account. Please note that several disconnected pieces of each type might exist because the exemplar $\mathcal{S}$ can be of arbitrary topology and some types of pieces might be missing altogether. Pieces of different type have a different purpose: types $r_{1,1}, r_{1,3}, r_{3,1}$, and $r_{3,3}$ form the "corner stones" that are instantiated exactly once. The pieces $r_{1,2}, r_{2,1}, r_{2,3}, r_{3,2}$ are replicated in one direction each, forming the boundary of the grid. Finally, the $r_{2,2}$ pieces are replicated in two directions.

Identifying tileable grids: Not all pairs of conflicting 1-grids create feasible 2-grids because the inner regions $r_{2,2}$ can have nontileable boundaries. This can happen because a pair of symmetric primary/secondary docking sites is cut into pieces by a different pair of docking sites. Although the pairs of curves are symmetric, the intersections do not need to be symmetric (Figure 7b). In order to obtain a valid tileable grid, we must demand two additional properties: First, the boundary curves of all regions $r_{i,j}$ must be symmetric under the transformations $\mathbf{T}_1, \mathbf{T}_2$. Second, the transfor-

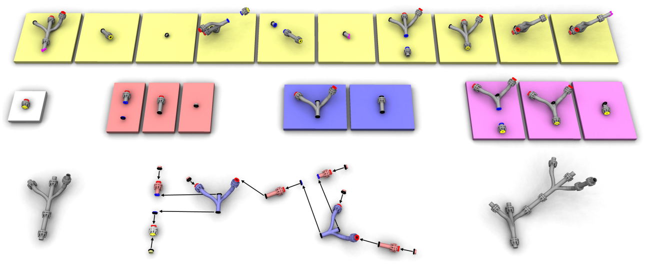


Figure 8: *Visualization of the grammar computed for the "pipe tree" example (Figure 10m). Each pad carries a docker, with the color indicating matching docking sites, shown as curves in the same color. Black curves indicate the sites at which the dockers are inserted into the parent docking site. The white docker is the root of the grammar. Below, an example assembly is given for illustration.*

mations $\mathbf{T}_1, \mathbf{T}_2$ must commute to obtain a well defined grid [Pauly et al. 2008]. If these two conditions are met, and we additionally assume that the inserted pieces do not collide with each other, it is easy to see that any instance of the grid is r-similar to the input surface. Our implementation currently checks all these conditions explicitly.

General case: The grid construction generalizes to the case of grids with k degrees of freedom: All simplices consisting of 2^k points need to match each other under the corresponding transformations, and all transformations need to commute. We can incorporate grids into the shape grammar by applying the grid detection algorithm to conflicting docking sites of the hierarchically extracted dockers.

4 Implementation

It is possible to perform all the computations described in the previous section directly on triangle meshes. However, this easily leads to robustness problems in practice due to sliver triangles. We avoid these issues by using an approximate representation: We store the symmetry information $\xi_r(\mathbf{T})$ only up to a fixed sampling resolution ϵ_s, in a voxel grid with uniform spacing ϵ_s. Each voxel that intersects with a piece of surface is labeled either symmetric or non-symmetric. We store this information compactly using a spatial octree. Although we employ a sampled representation to store symmetry information, the actual test for comparing geometry is still exact: We store the plane equations and boundary line equations of all triangles in each voxel and match them when comparing two pieces of geometry. We explicitly test for and exclude zero-area triangles where two edges are collinear up to numerical precision (which are unfortunately found frequently in real-world models). All computations of docking sites and dockers described in Section 3 are performed directly on the voxel grid representation. Intrinsic distances and neighborhoods are measured as graph distance in the graph of neighboring voxels, connected by a 26-neighborhood. Once we have identified the voxels that constitute docking sites and dockers, we use the boundaries of the voxels to cut out the docking site curves and dockers out of the original triangle mesh.

Discussion: The only approximation in this strategy is in the extend of the symmetric region, which might be underestimated by at most ϵ_s. We argue that this is a reasonable strategy: First, the algorithm will converge to an exact solution with shrinking ϵ_s, and thanks to the hierarchical representation, it is no problem to use an ϵ_s much smaller than r in practice. Furthermore, the similarity parameter r is usually only a vague guess by the user so that it does not seem necessary to approximate it with very high accuracy. In our examples, we always set ϵ_s to $r/4$.

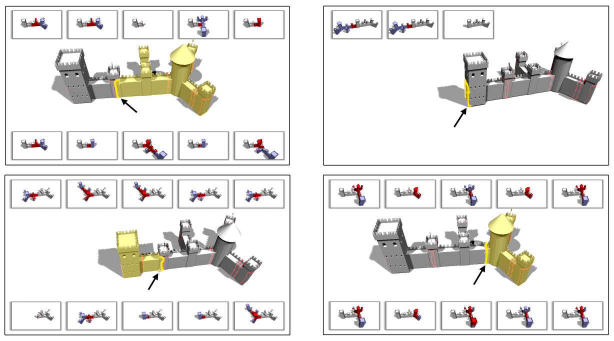


Figure 9: *Editing with context free grammars: possible edits for four different docking sites of the same model. The selected docking site and the docker to be replaced appear in yellow, other docking sites in light red. The rows of images above and below the model preview editing options. In these images, the red part is the new docker that will replace the yellow one. The blue parts are default geometry inserted to close the model; the user can exchange these next. Please note that many of the red dockers look similar but offer different sub-docking sites. In a context free grammar, these choices are encoded in different production rules if the alternative docking sites overlap. The accompanying video shows an interactive demonstration.*

4.1 Symmetry Detection

Our technique requires detecting symmetries in the input model as a first step [Mitra et al. 2006]. We use a variant of the algorithm of [Bokeloh et al. 2009], adapted to our voxel-based representation: Whenever a rigid motion maps the input surface $\mathcal{S}$ onto itself, we have found a partial symmetry. However, not all such symmetries are useful for modeling. For example, a planar area within an object can be partially mapped onto itself by a continuous set of infinitely many transformations. However, enumerating a large number of these will not be particularly useful for modeling. We therefore limit our algorithm to find "useful" symmetries by aligning salient feature lines [Bokeloh et al. 2009]. Next, we determine the area that is symmetric under each candidate transformation by computing the intersection of the regular voxel grid representation with a transformed version of itself. In each cell, we compare the original exact geometry, as described previously. This gives us a voxel-quantized approximation of $\xi(\mathbf{T})$. This strategy also works naturally with point cloud data (see [Bokeloh et al. 2009] for details). To obtain the r-symmetric set $\xi_r(\mathbf{T})$, we use a fast-marching algorithm to perform an erosion operation on the voxel grid. In the case of raw point-cloud data from 3D scanners, we add an additional processing step that removes isolated non-symmetric voxels within symmetric voxels in order to be robust to outliers.

4.2 Applications

We have implemented three example modeling tools within a prototypical inverse procedural modeling application.

Creating random shape variations: It is often useful to be able to create a large number of variations of a base geometry automatically, for example for creating background props in a game level or movie scene. We create random instance by executing random production rules from the computed shape grammar. For each rule, we check for collisions and revert to just using the original geometry of the docking sites if 10 random rules failed to work.

Semi-automatic modeling: We have also implemented an interactive editor to modify an existing shape according to user input (Figure 9). We start with the original model and display all docking sites as surface curves. The user can then "hover" over the model, docking sites are highlighted and the user can choose from the known production rules in order to change the model. Changes can be made at all levels of the hierarchy. The editor always displays a complete, r-similar model: When a new docker is inserted, its hierarchically dependent docking sites will be filled with the geometry that was originally contained in the docker, shown in a different color to indicate the default behavior.

Grid-based resizing: In addition to the context free rules used above, we also detect grid rules in the model. We let the user choose 1-grid or 2-grid rules and specify the number of repetitions.

5 Results

The results presented in this section are obtained from a single threaded C++ implementation of our framework running on a 2.6Ghz Core2 Duo computer with 8GB of RAM.

Grid-based editing: We have applied our method to models that contain grid structures. Examples for 1-grids are shown in Figure 10a and b, and 2-grids in Figure 10c and e-g. The spiral stairs scene (b) has complex micro-geometry below the handrail, which shows the numerical robustness of our implementation. The triangle mesh of the parking structure example (Figure 10g) contains many of non-manifold intersections, sliver triangles and doublet triangles that cover the same area, which is typical for models that have been designed primarily for rendering. Nevertheless, we obtain stable symmetry results.

Manual and random modeling: We have performed random example generation (Figure 10h,i and k-m) as well as manual modeling (Figure 10d and j). The random examples have been picked as typical examples out of a small number of random trials, all of which yielded reasonable geometry. An example grammar as computed by our algorithm is shown in Figure 8. The result is not as canonical as a manual, human designed grammar but simple enough to be useful in interactive manual modeling. With manual interaction, more control is possible. The editor (see Figure 9) is easy to use and greatly facilitates shape editing; the models shown in Figure 10 were assembled in less than a minutes each. The video accompanying this paper shows an interactive demonstration.

Point clouds: Figure 10h,l show results for scanned facade examples. For the front of the "new town hall", our algorithm has extracted 20 non-terminals corresponding to 40 dockers, all of which are of 1-grid type, due to the regular structure of the model. We use these rules to randomly insert and delete windows and the different tower elements of the facade, which lead to plausible results in all cases. Similar results are obtained for the "Zwinger" scan (courtesy of M. Wacker, HTW Dresden).

Analysis: Our input scenes have a complexity ranging from 1,040 (house, Figure 10e) to 500,000 (parking structure, Figure 10g) triangles. The new town hall facade consists of 1.2 million points. The computation times for our shape analysis are rather moderate, even for the large examples: For the facade, symmetry detection took 72 seconds, the computation of the docking sites and dockers 18 seconds, and building the grammar 125 seconds For the parking structure, the complete processing took 10 minutes, 2 minutes of which were spent on finding symmetries and extracting docking sites. This example is more time consuming because more symmetries can be found within an "exact" data set. Interactive editing and random example generation is performed in real-time, with response times (far) below one second for all examples shown in this paper. The effect of choosing parameter r is illustrated in Figure 10j: small values of r (middle row) produce piano keys with arbitrary grids, while a bigger value (bottom row) enforces the well-known 2/3 combinations of groups of keys. Our observation is that other than for such subtleties, the choice of r is not critical; all other examples use a fixed r that is set to 1.6% of the maximum bounding box side length of the object.

Limitations: Our current approach is limited to more or less exact symmetry and similarity, although some measurement noise or small inaccuracies are acceptable. However, we cannot handle any natural objects such as plants, animals, or people. Our current implementation is in addition limited to rigid similarity. With respect to shape grammars, our current applications are currently only using context free and grid-based rules, while context-sensitive modeling is still subject to future work. The rules that we extract do not cover the whole space of r-similar objects but only a subset; however, we can explicitly construct the members of this subset. In comparison to related texture-synthesis-based modeling approaches, we are not performing variational optimization. Therefore, we cannot "fit" models to arbitrary boundary conditions. In principle, the building blocks extracted by our algorithm could be used within a discrete MRF labeling algorithm to solve boundary value problems, however facing similar optimization problems as in texture synthesis. It is important to stress that texture/geometry synthesis *always* requires solving complex optimization problems, while our approach can still be used without.

6 Conclusions and Future Work

We have presented a theoretical framework for inverse procedural modeling of 3D objects and a practical implementation of a semi-automatic modeling system based on this framework. The main conceptual idea is to create a shape grammar that describes a large set of objects that are locally similar to a training exemplar under rigid motions. The key observation is that a grammar describing a large class of shape operations that maintain r-similarity can be directly derived from r-symmetry. The main algorithmic idea is the construction of docking sites and dockers: Whenever we can find a curve through a symmetric area that partitions the object into two pieces, we can derive a replacement operation that maintains r-similarity. A topological classification yields different types of operations (insert, delete, replace) and a hierarchical inspection of these operations then results in a context free grammar as well as grid-like replication rules. We believe that our proposal is only a first step into the mostly unexplored area of inverse procedural modeling, i.e., how to infer rules of how objects are build and structured solely from example instances. There are a number of open problems in our paper that we have to leave for future work: In particular, it would be interesting to generalize our framework to other notions of similarity like affine mappings with scaling (fractal patterns) or isometric mappings (bending invariant modeling). While most of our theoretical framework covers these cases, a practical evaluation still needs to be performed. In addition to this, it is still an open question how to define a shape grammar that includes all r-similar objects; for general input models our current construction covers only a subset. In a similar direction, it would also be interesting to evaluate in how far general, non-context free shape grammars can be used for shape modeling.

Acknowledgements

The authors would like to thank Alexander Berner, Qi-Xing Huang, Matthias Hullin, and Leonidas Guibas for useful discussions, and Gerd Wolf for creating many of our example models. We would also like to thank Ivo Ihrke, Tobias Ritschel, Qing Fang, and the anonymous reviewers for their feedback, helping substantially to improve the paper. This work has been partially supported by the Cluster of Excellence "Multi-Modal Computing and Interaction".

References

ALIAGA, D., ROSEN, P., AND BEKINS, D. 2007. Style grammars for interactive visualization of architecture. *IEEE Trans. Vis. Comp. Graph. 13*, 4, 786–797.

BARLA, P., BRESLAV, S., THOLLOT, J., SILLION, F., AND MARKOSIAN, L. 2006. Stroke pattern analysis and synthesis. *Computer Graphics Forum 25*, 3.

BHAT, P., INGRAM, S., AND TURK, G. 2004. Geometric texture synthesis by example. In *Symp. Geometry Processing*.

BOKELOH, M., BERNER, A., WAND, M., SEIDEL, H.-P., AND SCHILLING, A. 2009. Symmetry detection using line features. *Computer Graphics Forum 28*, 2.

CABRAL, M., LEFBVRE, S., DACHSBACHER, C., AND DRETTAKIS, G. 2009. Structure-preserving reshape for textured architectural scenes. *Computer Graphics Forum 28*, 2.

CHEN, L., AND MENG, X. 2009. Anisotropic resizing of model with geometric textures. In *Conf. on Geometric and Physical Modeling (SPM)*, ACM, New York, NY, USA, 289–294.

EFROS, A. A., AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. In *Proc. Int. Conf. Comp. Vision*.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by example. *ACM Trans. Graph. 23*, 3.

GAL, R., SORKINE, O., MITRA, N., AND COHEN-OR, D. 2009. iwires: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph. 28*, 3.

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

HART, J., AND FLYNN, O. C. P. 1997. Similarity hashing: A computer vision solution to the inverse problem of linear fractals. *Fractals 5*, 35–50.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proc. Siggraph 2001*, 327–340.

IJIRI, T., MĚCH, R., IGARASHI, T., AND MILLER, G. 2008. An example-based procedural system for element arrangement. *Computer Graphics Forum 27*, 3.

KRAEVOY, V., JULIUS, D., AND SHEFFER, A. 2007. Shuffler: Modeling with interchangeable parts. In *Pacific Graphics 2007*.

KRAEVOY, V., SHEFFER, A., SHAMIR, A., AND COHEN-OR, D. 2008. Non-homogeneous resizing of complex models. *ACM Trans. Graph. 27*, 5, 1–9.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph. 22*, 3, 277–286.

LAGAE, A., DUMONT, O., AND DUTRÉ, P. 2005. Geometry synthesis by example. In *Conf. Shape Modeling and Applications*.

LAI, Y.-K., HU, S.-M., GU, D. X., AND MARTIN, R. R. 2005. Geometric texture synthesis and transfer via geometry images. In *Symp. Solid and Physical Modeling*, 15–26.

MERRELL, P., AND MANOCHA, D. 2008. Continuous model synthesis. *ACM Trans. Graph. 27*, 5, 1–7.

MERRELL, P. 2007. Example-based model synthesis. In *Symp. Interactive 3D Graphics and Games*, 105–112.

MITRA, N. J., AND PAULY, M. 2008. Symmetry for architectural design. In *Advances in Architectural Geometry*, 13–16.

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.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3, 614–623.

MÜLLER, P., ZENG, G., WONKA, P., AND GOOL, L. V. 2007. Image-based procedural modeling of facades. *ACM Trans. Graph. 26*, 3.

NEUBERT, B., FRANKEN, T., AND DEUSSEN, O. 2007. Approximate image-based tree-modeling using particle flows. *ACM Trans. Graph. 26*, 3.

NGUYEN, M. X., YUAN, X., AND CHEN, B. 2005. Geometry completion and detail generation by texture synthesis. *The Visual Computer 21*, 9–10, 669–678.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *Proc. Siggraph 2001*, 301–308.

PAULY, M., MITRA, N. J., WALLNER, J., POTTMANN, H., AND GUIBAS, L. 2008. Discovering structural regularity in 3D geometry. *ACM Trans. Graph. 27*, 3.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1990. *The Algorithmic Beauty of Plants*. Springer Verlag.

RUSTAMOV, R. M. 2008. Augmented planar reflective symmetry transform. *Vis. Comput. 24*, 6, 423–433.

SHARF, A., ALEXA, M., AND COHEN-OR, D. 2004. Context-based surface completion. *ACM Trans. Graph. 23*, 3, 878–887.

SHARF, A., BLUMENKRANTS, M., SHAMIR, A., AND COHEN-OR, D. 2006. Snappaste: an interactive technique for easy mesh composition. *The Visual Computer 22*, 9, 835–844.

TAN, P., ZENG, G., WANG, J., KANG, S. B., AND QUAN, L. 2007. Image-based tree modeling. *ACM Trans. Graph. 26*, 3.

ŠT'AVA, O., BENEŠ, B., MĚCH, R., ALIAGA, D., AND KRIŠTOF, P. 2010. Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum*. to appear.

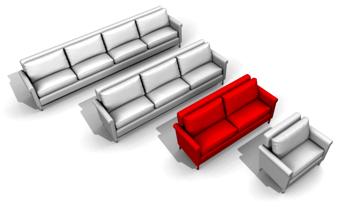
WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. Graph. 22*, 3, 669–677.

XIAO, J., FANG, T., ZHAO, P., LHUILLIER, M., AND QUAN, L. 2009. Image-based street-side city modeling. *ACM Trans. Graph. 28*, 5, 1–12.

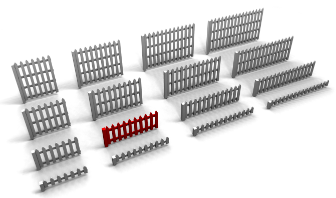
YEH, Y.-T., AND MĚCH, R. 2009. Detecting symmetries and curvilinear arrangements in vector art. *Computer Graphics Forum 28*, 2, 707–716.

ZELINKA, S., AND GARLAND, M. 2006. Surfacing by numbers. In *Graphics Interface 2006*.

ZHOU, K., HUANG, X., WANG, X., TONG, Y., DESBRUN, M., AND BAINING GUO, H.-Y. S. 2006. Mesh quilting for geometric texture synthesis. *ACM Trans. Graph. 25*, 3, 690–697.

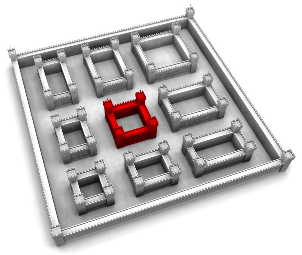

(a) sofas (1-grid)

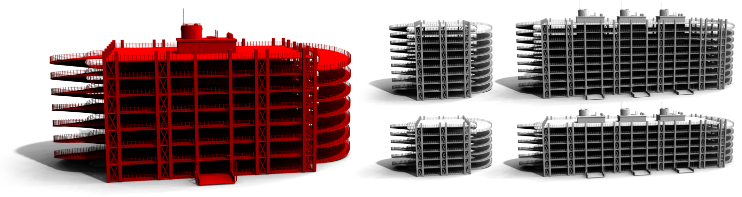
(b) spiral stairs (1-grid)

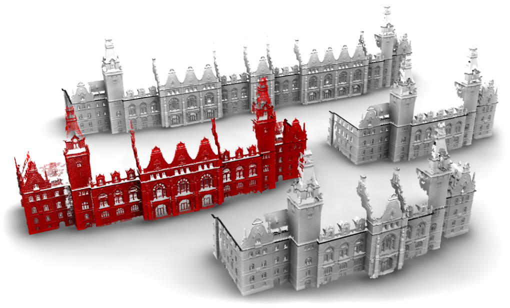

(c) fence (2-grid)

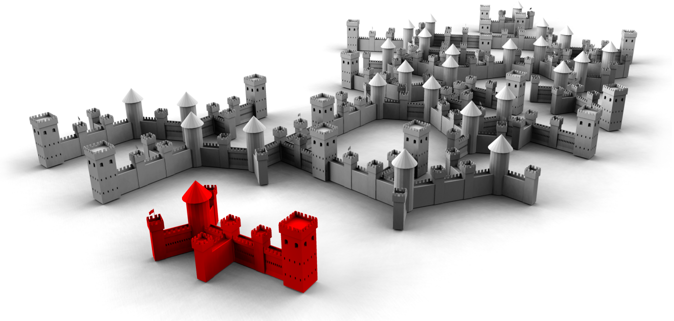
(d) bus station (ed)

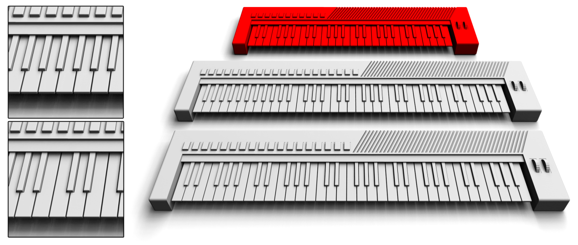
(e) house (2-grid)

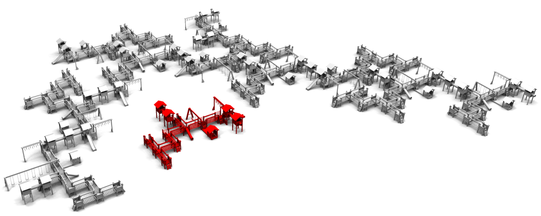

(f) castle (2-grid)

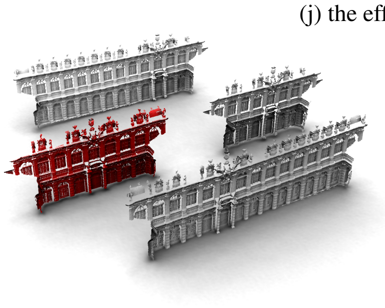

(g) parking structure (2-grid)
(from the Dosch Design shape collection)


(h) point cloud example: “new town hall” facade (rnd)
(courtesy of C. Brenner, IKG Hannover)


(i) castle (rnd)


(j) the effect of parameter r (ed)


(k) playground (rnd)


(l) point cloud: “Zwinger” (rnd)
(courtesy of M. Wacker, HTW Dresden)

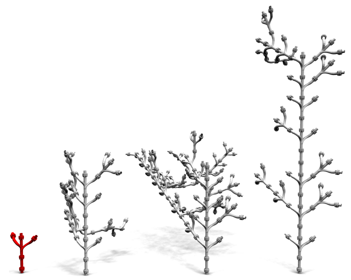

(m) pipe tree (rnd)

Figure 10: *Example scenes. The original scene is shown in red in each image. The scenes have been created by automatic random instancing (rnd), semi-automatic interactive editing (ed), or grid-resizing (k-grid). See the accompanying video for details.*