An Algebraic Model for Parameterized Shape Editing

Martin Bokeloh
Stanford University

Michael Wand
Saarland University and
Max-Planck-Institut Informatik

Hans-Peter Seidel
Max-Planck-Institut Informatik

Vladlen Koltun
Stanford University

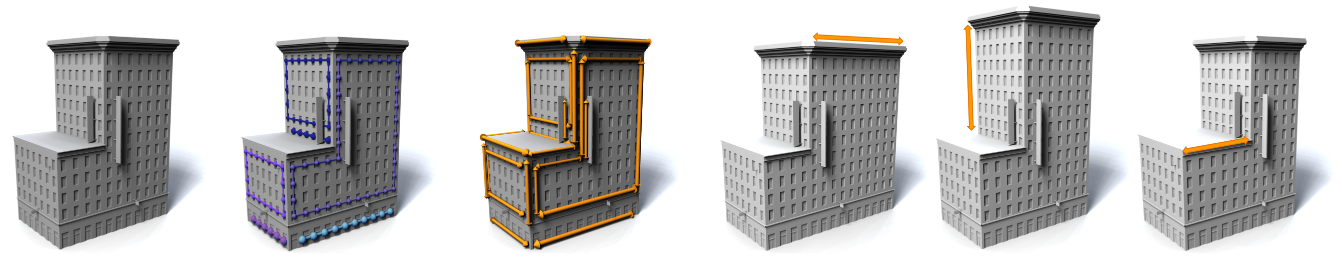


Figure 1: *Structure-adapting shape editing. An input shape (a) is automatically analyzed to extract regular translational patterns (b). Our algebraic model of shape regularity identifies useful degrees of freedom (c). These degrees of freedom are exposed for robust real-time shape editing (d).*

Abstract

We present an approach to high-level shape editing that adapts the structure of the shape while maintaining its global characteristics. Our main contribution is a new algebraic model of shape structure that characterizes shapes in terms of linked translational patterns. The space of shapes that conform to this characterization is parameterized by a small set of numerical parameters bounded by a set of linear constraints. This convex space permits a direct exploration of variations of the input shape. We use this representation to develop a robust interactive system that allows shapes to be intuitively manipulated through sparse constraints.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling;

Keywords: shape editing, shape understanding, symmetry, structural regularity

Links: DL PDF

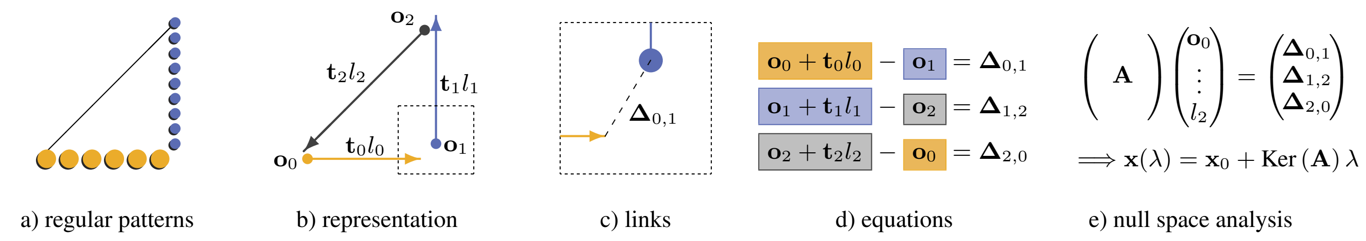
1 Introduction

Three-dimensional content creation is one of the key challenges of modern computer graphics. While we have sophisticated tools for displaying three-dimensional environments, the creation and editing of high-fidelity shapes needed to populate these environments is still a time-consuming and expensive task. One way to address this is to enable easy adaptation of existing shapes for new purposes. Towards this end, recent research has begun to explore structure-aware shape editing algorithms. Such methods define a structure model for a shape, extract according information from the input geometry, and use this knowledge to enable high-level shape manipulation.

A drawback of most existing approaches to structure-aware shape editing is the restriction to homeomorphic mappings: the topology of the object cannot be altered. The goal of our work is to develop a similarly robust and easy to use approach for *structure-adapting* shape editing, enabling intelligent adaptation of a shape's structure in response to high-level interactive manipulation. To this end, we introduce a new algebraic model of shape regularity. We observe that many objects, in particular man-made shapes, are composed of pieces of regular patterns: discrete patterns such as arrays of windows or ornamental elements, and continuous patterns such as planar surfaces and straight edges. Our model characterizes shapes as interlinked collections of such patterns.

The model is conceptually simple and is applicable to many man-made objects. It is specific to translational patterns, and the degrees of freedom represented in the model are limited to the translational generating directions of detected patterns (Figure 1). Each regular pattern is represented by a small set of variables and linear constraints, with geometric elements associated with some of the variables. Overlap and adjacency relationships between patterns lead to additional variables and linear constraints. The complete shape is represented by a linear system. The null space of this linear system defines the space of valid variations of the shape. A basis for the null space yields degrees of freedom for manipulating the shape. Interactive manipulation constraints placed by the user lead to a quadratic objective over the null space. The variation that conforms most closely to the interactive constraints is found by solving the resulting quadratic program.

In summary, the primary contribution of this paper is a new algebraic model that characterizes shapes in terms of interlinked regular patterns. We present a robust interactive shape editing system based on this model and demonstrate its effectiveness on shapes from publicly available repositories.



$\mathbf{o}_2$
$\mathbf{t}_2 l_2$
$\mathbf{t}_1 l_1$
$\mathbf{o}_0$
$\mathbf{t}_0 l_0$
$\mathbf{o}_1$
$\boldsymbol{\Delta}_{0,1}$
$\mathbf{o}_0 + \mathbf{t}_0 l_0 - \mathbf{o}_1 = \boldsymbol{\Delta}_{0,1}$
$\mathbf{o}_1 + \mathbf{t}_1 l_1 - \mathbf{o}_2 = \boldsymbol{\Delta}_{1,2}$
$\mathbf{o}_2 + \mathbf{t}_2 l_2 - \mathbf{o}_0 = \boldsymbol{\Delta}_{2,0}$
$\left(\mathbf{A}\right)\begin{pmatrix}\mathbf{o}_0\\ \vdots \\ l_2\end{pmatrix} = \begin{pmatrix}\boldsymbol{\Delta}_{0,1}\\ \boldsymbol{\Delta}_{1,2}\\ \boldsymbol{\Delta}_{2,0}\end{pmatrix}$
$\Longrightarrow \mathbf{x}(\lambda) = \mathbf{x}_0 + \mathrm{Ker}\,(\mathbf{A})\,\lambda$
a) regular patterns
b) representation
c) links
d) equations
e) null space analysis

Figure 2: *Analysis pipeline. (a) We analyze a shape for regular patterns and (b) represent each pattern as a triple* $(\mathbf{o}, l, \mathbf{t})$*, with origin* $\mathbf{o}$*, length/number of elements* l*, and a constant direction vector* $\mathbf{t}$*. (c,d) We link each pair of adjacent patterns by a linear equation that constrains the difference vector* $\boldsymbol{\Delta}_{0,1}$ *between the adjacent elements. In (c), the last element of pattern* 0 *is linked to the first element of pattern* 1*. (e) The space of valid solutions to the linear system is parameterized using the initial configuration* x_0 *and linear combinations of basis vectors for the null space of* $\mathbf{A}$*. This yields a low-dimensional representation of the degrees of freedom in the input shape.*

2 Related Work

In the last few years, structure-aware shape editing has gained a lot of attention. There have been continuous, topology-preserving deformation techniques and, more recently, techniques that allow for discrete modifications to the shape's structure.

Structure-aware deformation. Free-form deformation techniques are frequently employed to model organic shapes [Sederberg and Parry 1986; Sorkine et al. 2004; Botsch and Sorkine 2008]. However, working on complex man-made objects remains challenging due to structural relationships that must be respected during editing. Kraevoy et al. [2008] propose the first method for content-aware resizing of geometric models. The method enables axis-aligned stretching and protects "vulnerable" regions by adapting the elasticity. Cabral et al. [2009] constrain mesh deformations to preserve edge orientations in a least-squares sense. Rather than modeling algebraic regularity of the geometry, they focus on adapting textures when the shape is edited. Yang et al. [2011] explore shape variations under nonlinear constraints motivated by architectural applications but parametrize the shape space only locally using Taylor approximations. Xu et al. [2009] create a joint-aware deformation model based on slippage analysis. Gal et al. [2009] use symmetry constraints to maintain the global structure of the model under deformation. These structure-aware deformation techniques can substantially reduce the amount of user interaction required to create plausible shape variations. Nevertheless, all of these techniques are restricted to deformations with fixed topology.

Discrete reassembly. A promising approach to producing shape variations is inverse procedural modeling [Aliaga et al. 2007; Št'ava et al. 2010; Bokeloh et al. 2010]. Such methods often rely on inferring a shape grammar from the input shape. One limitation of this approach is controllability. While forward modeling is easy, finding a production of a shape grammar that fits user constraints remains a difficult combinatorial problem [Talton et al. 2011]. Our approach is different: we represent the space of valid variations as the feasible region of a linear system. Finding a variation that fits user constraints reduces to optimizing a quadratic objective over this region. Our technique is based on structural regularity detection as introduced by Pauly et al. [2008]. A subsequent paper by the same authors applied this model to shape editing [Mitra and Pauly 2008]; however, the paper does not consider complex relationships between multiple regular patterns.

The most closely related work to ours is by Bokeloh et al. [2011]. This prior work is based on content-aware elastic deformation, building on the techniques of Kraevoy et al. [2008]. The prior approach detects 1-parameter patterns in the input and uses "sliding dockers" to adaptively insert or remove pattern elements. Our work follows a similar line of thought, but our approach is both simpler and more general. The combination of continuous variational deformation and discrete algebraic regularity developed in Bokeloh et al. [2011] is unsatisfying: this approach does not explicitly model the pattern structure of the object but rather uses elastic deformation to adjust patterns locally. Furthermore, although the approach uses fairly complex finite-element deformation, the least-squares formulation does not yield exact results and can retain significant residual deformation errors due to the trade-off between different energy terms.

In contrast, our approach explicitly characterizes the space of valid variations through a set of simple algebraic conditions. Our model can handle multi-parameter grids and their interactions. It avoids the substantial deformation artifacts of the previous approach and produces clean shapes with no residual bending, even with quick and imprecise interactive manipulation. It is also much more computationally efficient, allowing rapid interactive exploration.

Another related recent work is by Lin et al. [2011], who retarget irregular architectural models in three orthogonal directions based on a user-guided hierarchical box decomposition. Our approach is different in that we construct the structure representation fully automatically, yielding an algebraic regularity model that does not rely on user-provided constraints. Furthermore, our work supports a broader range of variations, not limited to axis-aligned resizing.

3 Algebraic Model

We denote the input surface by $\mathcal{S} \subset \mathbb{R}^3$. We assume that $\mathcal{S}$ is a triangle mesh of arbitrary topology. Our goal is to describe the structure of $\mathcal{S}$ and to represent the space of valid variations of this shape.

Our structure model is based on regular patterns. A regular pattern represents a part of the input surface compactly by replicating a geometric element in a regular fashion. Manipulation of a single regular pattern is fairly easy since it only requires changing independent parameters (such as moving the origin or changing the number of elements). However, even mildly complex shapes contain regular patterns that interact with each other, such that their parameters can no longer be manipulated independently. The key idea of our approach is to detect regular patterns in $\mathcal{S}$, analyze how these patterns interact to form the overall composite shape, and identify possible degrees of freedom that maintain the important relationships between the different patterns.

First, we decompose the entire input shape into regular patterns. The result of this step is a set of geometric elements and associated parameters. In the following, we will give a precise definition of regular patterns (Section 3.1) and discuss how to resolve their ambiguities (Section 3.2). The actual extraction process is based on algorithms developed in prior work and is discussed in Section 5.1. After pattern analysis, we investigate the interaction of patterns and set up linear equations that constrain neighboring patterns in order to maintain important pairwise relationships (Section 3.3). This results in an underconstrained linear system, the null space of which yields a direct parameterization of the degrees of freedom in the input shape (Section 3.4). A schematic overview of the analysis is provided in Figure 2.

3.1 Regular Patterns

Regularity in shapes appears in many different forms. In this work, we consider the type of regularity induced by a set of generator transformations $\{T_1, \ldots, T_k\}$ that generate a symmetry group $\mathcal{G} = \{T_1^{i_1} \circ \cdots \circ T_k^{i_k} | i_1, \ldots, i_k \in \mathbb{Z}\}$. We restrict the generator transformations to be translations. For a detailed introduction please refer to [Pauly et al. 2008].

Typically, a single regular pattern covers only a small part of a shape whereas symmetry groups describe global structures. Consequently, we consider regular patterns as structures with limited spatial support that we express by an element $\mathcal{E} \subset \mathcal{S}$ that is replicated using transformations of a finite subset $\mathcal{G}_P \subset \mathcal{G}$. We define a regular pattern as $\Pi = (\mathcal{G}_P, \mathcal{E})$. In the following, we will describe different types of regular patterns and their parameterization.

Discrete 1-parameter pattern ("line patch"): A discrete line patch is a regular pattern with one generator transformation that translates an element $\mathcal{E}$ at least three times (Figure 3b). We parameterize a line patch by $(\mathbf{o}, l, \mathbf{t})$, where $\mathbf{o}$ is the origin of the pattern (centered at the first element), l refers to the "length" of the pattern which corresponds to the number of elements minus one, and $\mathbf{t}$ is the generator translation that will stay constant. Using this parameterization we can translate the whole pattern by moving the origin or resize the pattern by changing the number of elements. Importantly, we cannot produce negative elements, so we constrain each length variable to be nonnegative: $l \geq 0$.

Discrete 2-parameter pattern ("area patch"): Discrete area patches are subsets of regular grids that we describe by specifying polygonal boundaries (Figure 3c). Each edge of a boundary polygon is modeled as a discrete line patch where each individual generator transformation is an integer combination of the two generators that span the grid.

We do not consider discrete 3-parameter patterns here since they are not very common in practice. However, the generalization to this case would be straightforward by using discrete area patches to enclose a region of a volumetric 3-parameter grid.

Continuous pattern: Both previous cases have continuous counterparts (Figure 3d,e). A continuous area patch is a polygon obtained directly from the input mesh where each boundary edge is represented as a continuous line patch. We consider polygon edges as fixed if their edge length is below a threshold r (we use r=1% of the diameter of $\mathcal{S}$). A continuous line patch is a special case of a continuous area patch where some edges are fixed, such as the elongated quads shown in Figure 3d.

Mixed discrete/continuous pattern: In some cases a discrete line patch and a continuous line patch form a mixed-case area patch (Figure 3f).

0-parameter pattern ("rigid patch"): A rigid patch is a piece of geometry without regular structure. This means that all polygon

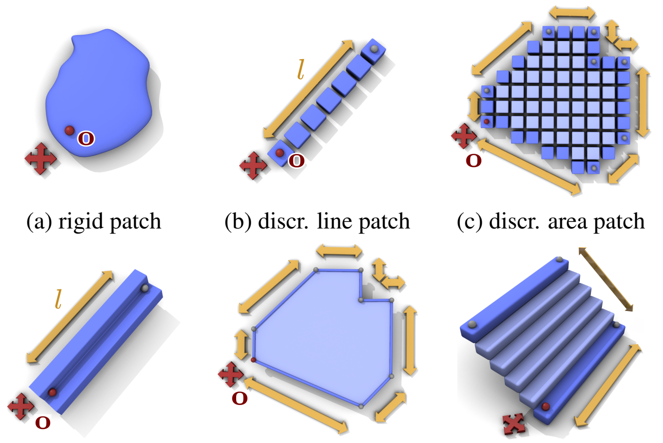


Figure 3: *Different types of regular patterns and their parametrization. (a) Each patch has an origin* $\mathbf{o}$*. (b) Discrete line patches add a length variable* l*. (c) The boundary of an area patch is defined by a continuous polygon, which is represented as a collection of line patches that are coupled to the area vertices. (d,e) Line patches and area patches can be continuous. (f) Discrete and continuous line patches can form a mixed-case area patch.*

edges are too small to be considered as continuous line patches ($l < r$). In this case we parameterize the rigid patch by the position of its origin o (Figure 3a).

3.2 Normalization

The previous definition of regular patterns permits a large number of different representations of one and the same regular structure. We therefore normalize the representation.

Maximal patches: Larger patterns usually contain an exponential number of subsets that also form regular patterns. Therefore, we restrict our model to maximal patterns. If there exist $\Pi_1 = (\mathcal{G}_{P1}, \mathcal{E})$ and $\Pi_2 = (\mathcal{G}_{P2}, \mathcal{E})$ with $\Pi_1 \subset \Pi_2$, we only keep the second structure. Among these, we maximize the area, i.e., use the pattern with maximal $\mathcal{E}$. If different patterns can explain the same geometry, we keep the one with the largest number of elements. If the same geometry can be described by different patterns with an identical number of instances, we keep an arbitrary pattern.

If one pattern is entirely contained in a grid element of another pattern, we use only the largest pattern with respect to set inclusion. We currently do not model hierarchical nesting of patterns.

3.3 Linked Patterns

In the following, we assume that the input shape $\mathcal{S}$ has been decomposed as a union of potentially overlapping regular patterns $\mathcal{S} = \Pi_1 \cup \ldots \cup \Pi_n$, including rigid patches that represent non-repeating geometry. We proceed in two steps. First, we detect pairwise adjacencies and intersections of patterns, forming an undirected graph that links interacting patterns. Afterwards, for each such link $\{\Pi_i, \Pi_j\}$, we create a set of linear constraints on its variables that maintain the link structure.

Computing the link graph: The link graph is established straightforwardly by checking all pairs of regular patterns for intersection or adjacency. We link those pairs of patterns whose distance is smaller than a constant ϵ that corresponds to modeling accuracy.

Link constraints: Having determined the link graph, we iterate over all links and create a set of constraints that couple corresponding patterns. The basic idea is that we couple individual *elements* of a pair of patterns by constraining the difference of their relative position with a linear equation, where we use, as before, the element centers as (arbitrary) reference points (see Figure 4).

We now make this notion of coupling more precise. For simplicity, we first restrict ourselves to line patches; area patches are discussed subsequently. To simplify the notation, we refer to the center of an element i in line patch Π_a by $\mathbf{e}_i(\Pi_a)$, where the index refers to the original configuration. The center of such a pattern element can be expressed by a linear combination of pattern origins and length variables. For example, the center of the first element in pattern a is denoted by $\mathbf{e}_{min}(\Pi_a) = \mathbf{o}_a$. Correspondingly, the center of the last element of a line patch Π_a can be computed as $\mathbf{e}_{max}(\Pi_a) = \mathbf{o}_a + \mathbf{t}_a l_a$. For in-between values, we use extra variables, as explained below.

Let $\mathbf{e}_i(\Pi_a)$ be an element of pattern Π_a and let $\mathbf{e}_j(\Pi_b)$ be an element of pattern Π_b. A linear constraint that encodes a link between these two elements has the following form:

$$\mathbf{e}_i(\Pi_a) - \mathbf{e}_j(\Pi_b) = \Delta^{i,j}_{a,b}, \qquad (1)$$

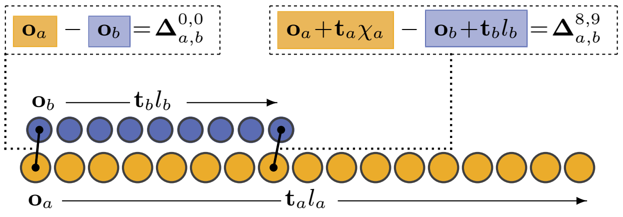
where $\Delta^{i,j}_{a,b}$ is the difference vector between the two elements in the original shape. For example, in Figure 2c we couple the last element of pattern Π_0 to the first element of pattern Π_1.

Extra variables: The patches in our model are fully described by variables $(\mathbf{o}_0, \ldots, \mathbf{o}_n, l_0, \ldots, l_m)$ that specify pattern origins and lengths. In order to reason about the constraints imposed upon in-between elements, we introduce additional variables of the form χ^i_P, where χ^i_P refers to the i-th element of pattern P. (The index i refers to the value of χ^i_P in the original shape. The actual value of χ^i_P can (and generally is) different from its original value i when the shape is manipulated.) These extra variables are created as needed. A variable is only created when it is needed to define a link, and only once in the case of multiple links that involve the same element.

To maintain the semantics of elements within a pattern, we constrain their domain. For a line patch, we add the linear inequalities $0 \leq \chi^i_P \leq l_P$. If multiple extra variables are defined within a pattern, we also add linear constraints that preserve their original ordering: $\chi^i_P \leq \chi^j_P$ for $j > i$.

Area patches: Area patches are handled analogously. In case of linking to an element along the boundary, we use the already established mechanism for the line patches. For inner vertices, we create extra variables that are constrained to remain within the area of the original polygon. In our current implementation, we approximate this constraint, which in general is non-convex, with a simplified convex constraint. Instead of the full polygon area, we permit only the intersection of all half-spaces the point considered resides in. This is a conservative approximation: it guarantees that we always obtain valid polygons, without intersections or links outside the polygon, but the maneuverability of the element is in general reduced.

So far, we have discussed how to couple a pair of elements. Depending on the type of the two adjacent patterns and their interaction, we can obtain a varying number of such element-wise constraints. In the following, we will describe how the actual constraints are computed when different types of patterns interact.



$\mathbf{o}_a$ − $\mathbf{o}_b$ = $\boldsymbol{\Delta}^{0,0}_{a,b}$
$\mathbf{o}_a + \mathbf{t}_a\chi_a$ − $\mathbf{o}_b + \mathbf{t}_b l_b$ = $\boldsymbol{\Delta}^{8,9}_{a,b}$
$\mathbf{o}_b$
$\mathbf{t}_b l_b$
$\mathbf{o}_a$
$\mathbf{t}_a l_a$

Figure 4: *A link between two line patches. In order to couple the last element of pattern b to an element of pattern a, we introduce a new variable χ_a. In the initial shape, $\chi_a = 8$. The vector $\Delta^{8,9}_{a,b}$ is the offset between the two elements in the initial configuration.*

(1-1)-interaction, line to line patch: If two line patches Π_a, Π_b are linked, we first consider their generating translations $\mathbf{t}_a, \mathbf{t}_b$. If they are collinear, the configuration corresponds to overlapping intervals (Figure 4). Otherwise, they intersect at a point. In the latter case, we create one constraint that couples the pattern elements closest to the intersection point. As discussed before, we might add new variables here to identify interior elements. In the former case, we create two constraints that bound the interval that the two patterns have in common. This case is illustrated in Figure 4.

(1-2)-interaction, line to area patch: Again, the linear pattern can be coplanar to the area patch or intersecting. An intersection leads to one constraint that couples the elements closest to the intersection point. Coplanarity leads to two constraints that couple the beginning and end of the overlapping interval.

(2-2)-interaction, area to area patch: Two coplanar area patches are linked by coupling the elements closest to the intersection points of the boundaries of the patches. Non-coplanar area patches intersect in auxiliary line patches. These are handled as in the (1-1)-interaction case. Most prevalent in typical models is the link of adjacent area patches at boundaries, which is handled as a (1-1)-interaction.

(0-1)- and (0-2)-interactions with rigid patches: Rigid patches remove degrees of freedom. As in the previous cases, we create constraints that link the origin of the rigid pattern to the intersection line or surface.

3.4 Null Space Analysis

The link constraints from the previous section form a large, heavily redundant linear system:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \qquad (2)$$

where $\mathbf{x}$ is the vector of pattern origins, pattern lengths, and extra variables. The vector $\mathbf{b}$ is composed of the offsets that were observed in the original configuration. In order to preserve these relationships we keep $\mathbf{b}$ constant. Typically, the linear system is underdetermined and yields a linear subspace of solutions. By analyzing the kernel of matrix $\mathbf{A}$ we can directly parameterize this subspace. Let $\mathbf{K}_A$ be a basis for the linear subspace Ker$(\mathbf{A})$. We can now express all possible solutions as a function of a parameter vector λ:

$$\mathbf{x}(\lambda) = \mathbf{x}_0 + \mathbf{K}_A \lambda. \tag{3}$$

Each column of $\mathbf{K}_A$ points in a direction that corresponds to changing a set of pattern parameters while still maintaining the pattern relationships. The vector λ parameterizes these changes. To produce solutions for equation (2) we need to translate the linear subspace by an arbitrary solution x_0 (in this work, we simply choose the initial state). We call $\lambda \in \mathbf{\Lambda}$ the parameter representation of a shape and $\mathbf{\Lambda}$ the parameter domain.

Inequalities: Not all solutions to equation (3) correspond to valid shapes because only a subset meet the inequalities that constrain the original variables. These inequalities include nonnegativity constraints on pattern lengths and extra variables, and ordering constraints on the variables. We project all inequalities into the null space, which allows us to perform all computations in this low-dimensional space. Let $\mathbf{Mx} \geq \mathbf{m}$ be all inequalities. Expressing the inequalities in parameter domain yields:

$$\underbrace{[\mathbf{M}\,\mathbf{K}_A]}_{=\mathbf{M}'} \lambda \geq \underbrace{\mathbf{m} - \mathbf{M}\mathbf{x}_0}_{=\mathbf{m}'} . \tag{4}$$

Choosing a basis for the null space: In simple cases, the kernel of $\mathbf{A}$ can be computed by row reduction techniques such as Gaussian elimination. These approaches are numerically unstable and therefore not advisable in general, however we use them to remove trivial redundancy. We iteratively apply three basic types of operations: remove constant variables, remove identical rows, and substitute variables by other dependent variables. However, we do not perform substitutions if they are not well-conditioned, i.e., if the omitted variable cannot be stably computed from the substitution.

A numerically stable method to compute the null space is based on the singular value decomposition (SVD). Let $\mathbf{A} = \mathbf{U\Lambda V}^T$ be the decomposition. A basis $\mathbf{K}_A$ for the null space is formed by the columns of $\mathbf{V}^T$ that are associated with singular values that are equal to zero. To compensate for numerical noise and small inaccuracies in the linear system we consider singular values to be zero if their value is below 0.01% of the largest singular value.

4 Shape Editing

The algebraic model defined in the previous section provides a direct parameterization of the degrees of freedom in the shape and a set of inequalities that define the set of valid parameters. In this section, we utilize this model for shape editing. First, we describe individual types of interactive constraints used in our shape editing system, which lead to a quadratic objective over the parameter space. Then, we formulate the complete quadratic program.

4.1 Interactive Constraints

Our implementation supports two types of interactive manipulation constraints: point constraints and difference constraints. For point constraints, the user selects a pattern element and drags it to a specific target point y. We select the pattern origin $\mathbf{e}_{closest}$ closest to the selection point and formulate a quadratic objective term:

$$E_{point}(\mathbf{x}) = \left(\mathbf{e}_{closest} - y\right)^2 . \tag{5}$$

Difference constraints are defined similarly. The user selects two pattern elements $\mathbf{e}_{c_0}$ and $\mathbf{e}_{c_1}$, and specifies their difference vector δ_{c_0,c_1}. The corresponding objective term is defined to be

$$E_{diff}(\mathbf{x}) = \left(\mathbf{e}_{c_0} - \mathbf{e}_{c_1} - \delta_{c_0,c_1}\right)^2 . \tag{6}$$

4.2 Regularization

With only a few sparse user constraints, the optimization problem stated in Equation 9 is still underconstrained and admits many unreasonable solutions. We therefore introduce a regularization objective that aims to keep the original values of the length variables. This regularization term is weighted very weakly in comparison with the interactive constraints. Let l_i be a length variable of pattern i and l_i^0 be its original value. The regularization objective is defined as

$$E_{reg}(\mathbf{x}) = w_{reg} \sum_i \left(l_i - l_i^0\right)^2 . \tag{7}$$

4.3 Quadratic Program

We sum all of the above terms into a quadratic objective:

$$\begin{aligned} E(\mathbf{x}) &= E_{point}(\mathbf{x}) + E_{diff}(\mathbf{x}) + E_{reg}(\mathbf{x}) \\ &= \mathbf{x}^T \mathbf{Q}\,\mathbf{x} + \mathbf{x}^T \mathbf{q}. \end{aligned}$$

This objective is formulated in the original domain and then projected into the parameter domain:

$$E'(\lambda) = \lambda^T \left((\mathbf{K}_A)^T \mathbf{Q}\,\mathbf{K}_A \right) \lambda + \lambda^T \left(\mathbf{K}_A \mathbf{q}\right) . \tag{8}$$

Overall, we minimize the quadratic objective (8) subject to the inequalities (4) in the parameter domain:

$$\begin{array}{ll} \text{Minimize} & E'(\lambda) \\ \text{subject to} & \mathbf{M}'\lambda \geq \mathbf{m}'. \end{array} \tag{9}$$

A significant advantage of this formulation is that it separates the modeling of the space of valid variations of the shape, which is characterized through linear equations and inequalities, from the interactive constraints and the regularization term, which are expressed as quadratic objectives. User constraints are typically somewhat imprecise. The same is true for the regularization term, which defines a general default behavior for the shape. On the other hand, modeling the space of valid shapes requires precise alignment of adjacent patterns in order to avoid gaps or holes. If we were to formulate this with strong quadratic constraints we would end up with conflicting objectives where shape validity is traded off against imprecise user input. In our approach, the interactive constraints do not compromise the correctness of the shape (Figure 5).

5 Implementation

5.1 Pattern Detection

In order to implement the framework described in the preceding sections, we still need a technique for detecting symmetries and regular grids in 3D shapes. Our implementation follows the approach of Bokeloh et al. [2011], which is based on the RANSAC technique. We randomly sample generator transformations from mesh features and evaluate the number of elements that belong to the pattern defined by the generator. This is repeated a number of times (up to 500 in our implementation) and only the dominant pattern (with the greatest number of elements) is kept and removed from the sampling set. The whole process is iterated until no patterns can be found. The remainder of this section describes this process in more detail.

Pattern primitives and cascaded detection: An important difference from the implementation described by Bokeloh et al. [2011] is that we do not cut triangles. Our method operates directly on the mesh structure and uses three types of pattern primitives: connected components, polygons, and polygon edges. We apply our detection techniques in a top-down manner. First, we perform pattern detection on connected components only and remove all elements that belong to a pattern. Then, we extract all polygons from the remaining set and repeat the same process on the polygon level. Polygon extraction is done by clustering adjacent triangles with similar normal vectors into polygons, which need not be convex. Finally, we search for patterns within polygons by considering edges as primitives. This top-down approach is not only faster than non-hierarchical methods but also produces better results. The connected components often carry important information from the modeling process of the original shape. Additionally, detecting discrete boundary patterns of polygon edges avoids heuristic cutting of triangles into repeating elements and yields a persistent representation that maintains mesh quality.

For the following detection modules we equip each pattern primitive with a small descriptor that consists of the centroid $\mathbf{d}_c$, the radius d_r of the bounding sphere centered at $\mathbf{d}_c$, and the number of vertices d_n associated with that primitive. We cluster primitives based on d_r and d_n into classes and perform pattern detection within these classes of primitives. Polygon edges are not shared across different polygons. We observed that additional classification of polygons based on their parent (a connected component) yields more meaningful decompositions in practice.

Discrete 1-parameter pattern detection: We randomly select two primitives (referred to as first and second sample) and define a line through their centroids. Then, we gather all primitives with centroids close to the defined line. In this subset we again perform random sampling to find generators with the maximal number of consecutive elements. In order to be robust against imprecisely placed elements with slightly varying generators, we allow the elements to be misplaced by 3% of the generator's length. We also need to make sure that all elements have the same geometry and the same orientation. Therefore, we compare each primitive with the first selected primitive before adding it to the subset. To compare two primitive we align both by aligning their centroids and test if each vertex has a counterpart within modeling accuracy ϵ.

We repeat this process several times and extract only the pattern with the highest number of elements. For each detected pattern we mark each associated primitive and prohibit a new pattern from starting with one of these primitives. (However, the second sample can still be drawn from the set of marked primitives.) This ensures that we extract a pattern only once but still allow patterns to share elements. To improve the recognition performance we use an octree-based sampling strategy that draws the second sample in proximity to the first one.

Discrete 2-parameter pattern detection: For 2-parameter patterns we use a similar sampling approach to the 1-parameter case. We select three primitives, create a plane, and gather primitives near that plane. In contrast to the 1-parameter case, instead of randomly sampling two generator transformations from the subset, we apply a generator voting technique first to retrieve a set of possible generators [Pauly et al. 2008]. Then, we randomly sample from this set of generators to obtain area patches where we choose the area patch with the highest number of elements. Since 1-parameter patterns are included in 2-parameter patterns, we search for area patches before searching for line patches.

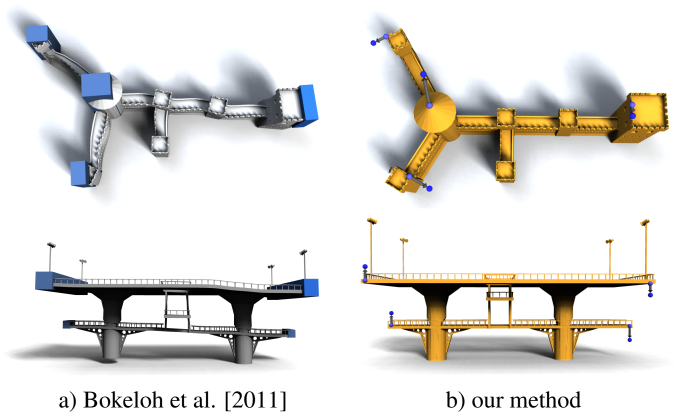


Figure 5: *Badly placed interactive constraints force the prior technique of Bokeloh et al. [2011] to distort the shape. In contrast, our method always produces a valid shape that is as close as possible to the interactive constraints.*

5.2 Instantiating Models

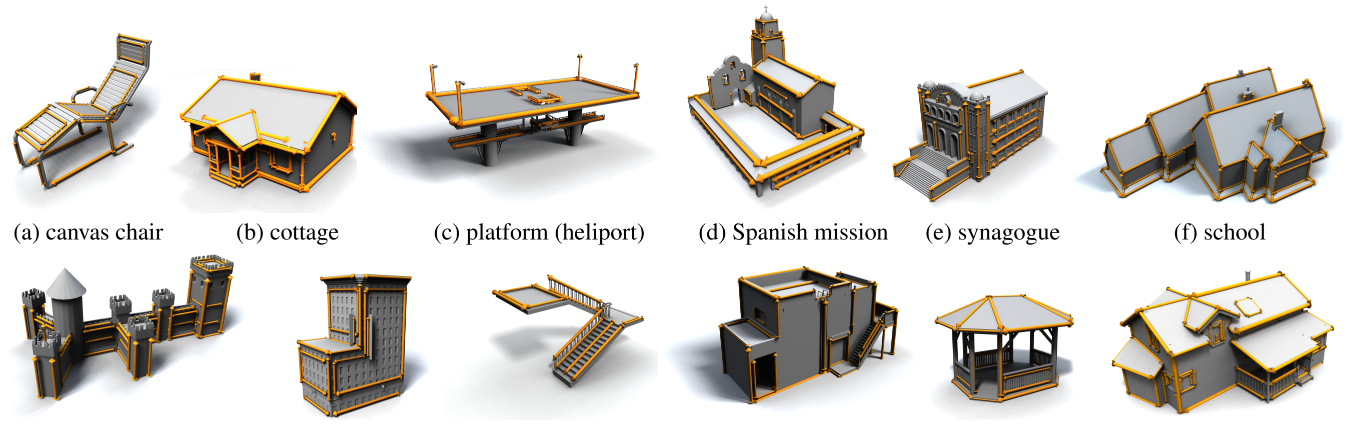
Producing the mesh that corresponds to the manipulated shape is easy with our representation. For discrete patterns we round the number of elements to the next integer, create the desired number of elements, and apply affine scaling to match the given pattern length. Discrete area patches are rasterized without scaling. Complex polygons, for example with a discrete pattern along the boundary, are triangulated using a standard quadtree-based polygon meshing algorithm [de Berg et al. 2008].

6 Results

We have implemented a prototype shape editing system based on the representation and algorithms introduced in this paper. Our implementation is single-threaded. The experiments were performed on a consumer-grade workstation with a 2GHz Intel Xeon E5335 processor and 4GB of main memory. We use MOSEK for quadratic programming.

Editing software prototype: Our system permits the user to interactively manipulate shapes by placing handle constraints. In addition, it displays the available degrees of freedom, which are identified by projecting the indicator vectors that correspond to each length variable onto the parameter space and only retaining those whose projection is non-negligible. We visualize the degrees of freedom using arrows and perform farthest-point sampling to reduce visual clutter (Figure 6).

We have tested the system on a large number of shapes obtained from publicly available repositories, namely the Digimation Model Bank and the Google 3D Warehouse. Our test shapes and their degrees of freedom are shown in Figure 6. Data for each shape is provided in Table 1. Editing results are shown in Figure 7. The supplementary video demonstrates a number of interactive editing sessions, captured in real time.



(a) canvas chair (b) cottage (c) platform (heliport) (d) Spanish mission (e) synagogue (f) school

(g) castle (h) downtown hotel (i) stairs (j) modern house (k) pavilion (l) cabin

Figure 6: *Automated visualization of degrees of freedom for our test shapes.*

Our technique is able to extract meaningful degrees of freedom for all of the shapes. Typically, we obtain 25 to 200 degrees of freedom. The initial linear systems have between 10 and 100,000 equations and up to 50,000 variables. Incomplete elimination reduces the complexity to below 2000 variables in all cases. The complexity of the resulting parameter space depends strongly on the input model, but not necessarily on the mesh size: objects with many different pieces embedded in smooth polygons have more degrees of freedom than large, regular structures. For example, the modern house (example (j)) creates a complex parameter space due to many independent pattern origins.

We obtain plausible editing results for all test shapes and, equally important, our editing interface is easy to understand and work with. This is because we directly modify the main "variation modes" of the shapes, represented by numerical parameters rather than complex combinatorial rules. Preserving complicated combinations of regular patterns, for example the characteristic roof shapes in examples (b), (f), (k), and (l) can be quite challenging in traditional modeling tools. Our system alleviates the burden of detailed modeling and provides only solutions that obey the meaningful constraints imposed by the structure of the input shape. For example, when the user lifts the partial roof of the doorway in example (b), the method automatically resizes all dependent patterns, such as the fence below. However, the position and size of the entryway can still be controlled independently as shown in the magnified views in Figure 7(b). This and other examples are demonstrated in the supplementary video.

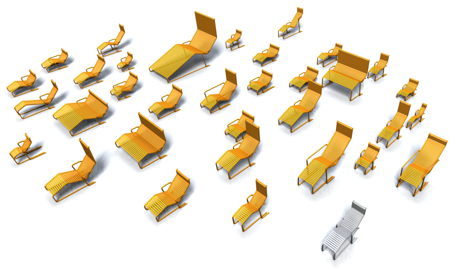
Performance: For most models, the automated analysis is performed in a few seconds; only some models with complex dependencies require more time. In these cases, the computation of the SVD, which currently uses the dense-matrix implementation of OpenCV, dominates the costs. Interactive shape optimization in response to user constraints runs at multiple frames per second, as shown in the video. Very large models can result in huge linear systems that we currently cannot handle due to large memory and run-time requirements of dense SVD.

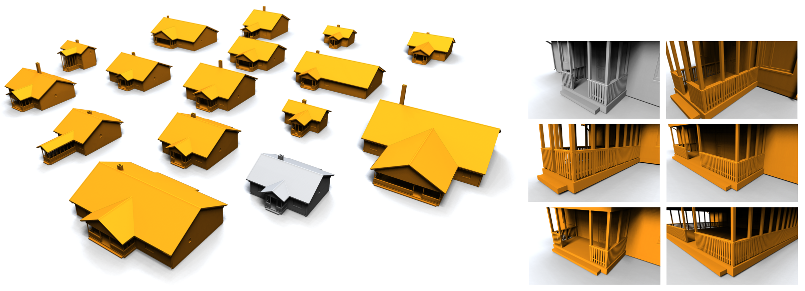
Comparison to prior work: In comparison to the previous approach of Bokeloh et al. [2011], we obtain a number of significant improvements. Our test shapes contain 2-parameter grids and overlapping grids, which cannot be handled by the previous approach. This is illustrated in Figure 8. Furthermore, our system extracts explicit numerical degrees of freedom, which has important practical implications. We have included the "canvas chair", "platform", and "castle" examples from the paper of Bokeloh et al. [2011] to illustrate the advantages of the algebraic approach. As shown in Figure 9 and in the video, the previous elastic deformation approach with least-squares structure constraints creates residual artifacts, which we avoid altogether. Furthermore, editing is automatically constrained to available degrees of freedom, which makes the interaction easier; one cannot accidentally "bend" the model when constraints move out of the feasible space as shown in Figure 5.

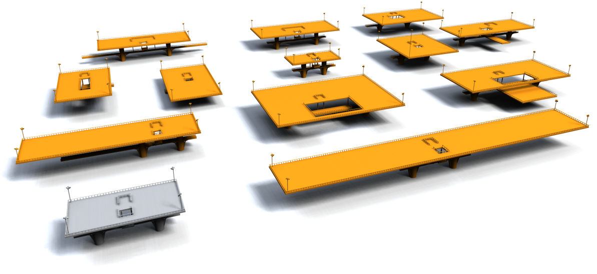
model	num. triangles	pattern detection	analysis
canvas chair (DM)	3232	263ms	925ms
cottage (DM)	8516	551ms	41s 895ms
platform (DM)	10132	822ms	1s 46ms
mission (DM)	18182	809ms	2s 440ms
synagogue (DM)	31644	4s 612ms	41s 237ms
school house (GW)	412	21ms	1s 577ms
castle	14370	3s 620ms	1s 470ms
downtown hotel (DM)	47475	19s	17s 800ms
stairs (GW)	4958	3s 526ms	947ms
modern house (DM)	3462	194ms	2s 632ms
cabin (GW)	3796	1s 256ms	5s 148ms
staircase (GW)	1728	1s 591ms	1s 982ms
pavilion (GW)	956	630ms	2s 103ms

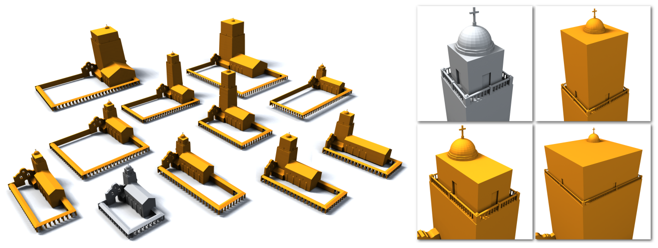
Table 1: *Data for our test shapes, including number of triangles and performance statistics (pattern detection time and time for setup and analysis of the quadratic program). The shapes were obtained from the Digimation Model Bank (DM) and the Google 3D Warehouse (GW).*

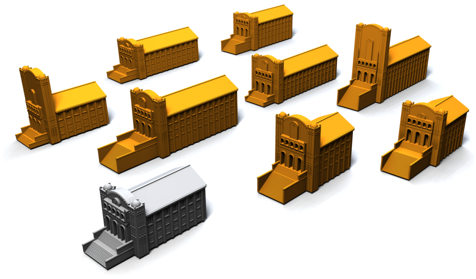
Limitations: Our method has a number of limitations. Our model is currently restricted to translational regular patterns, and can only handle rigidly symmetric parts, ruling out organic shapes. In some of the examples, artifacts can show up due to rounding. These can usually be removed by shifting constraints slightly. We have also not yet considered further modeling aids, such as maintaining nonregular and global symmetries in the spirit of Gal et al. [2009], which would be an interesting extension of our approach. Computationally, our method is limited in terms of structural complexity: while examples of moderately high complexity work very well, highly detailed geometry with many interleaving patterns bring our approach to its limits. This has two reasons. First, a large number of variables and equations causes problems with our current (dense)

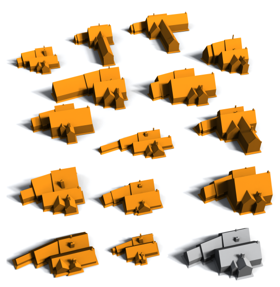

(a) canvas chair

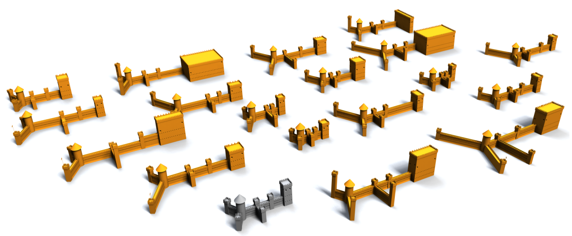

(b) cottage

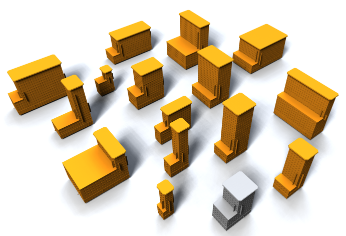

(c) platform (heliport)

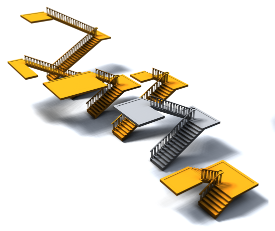

(d) Spanish mission

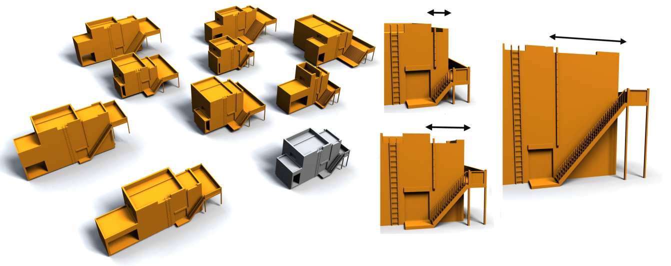

(e) synagogue

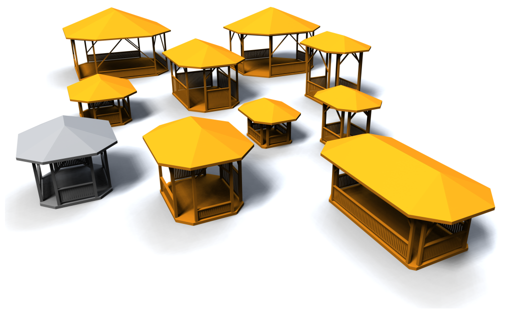

(f) school house


(g) castle


(h) downtown hotel


(i) stairs


(j) modern house


(k) pavilion

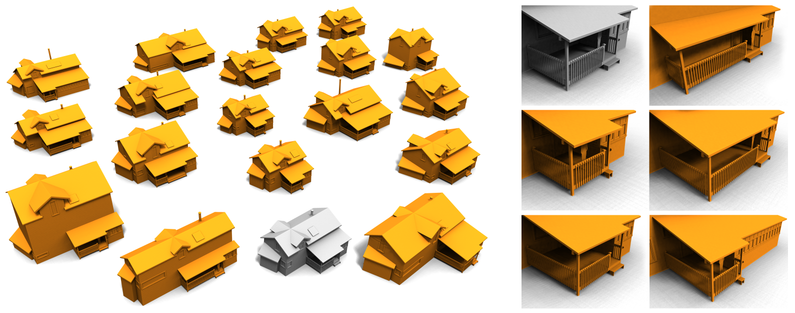

(l) cabin

Figure 7: *Interactive editing results (orange) for our test shapes (grey). Real-time editing sessions are shown in the supplementary video.*

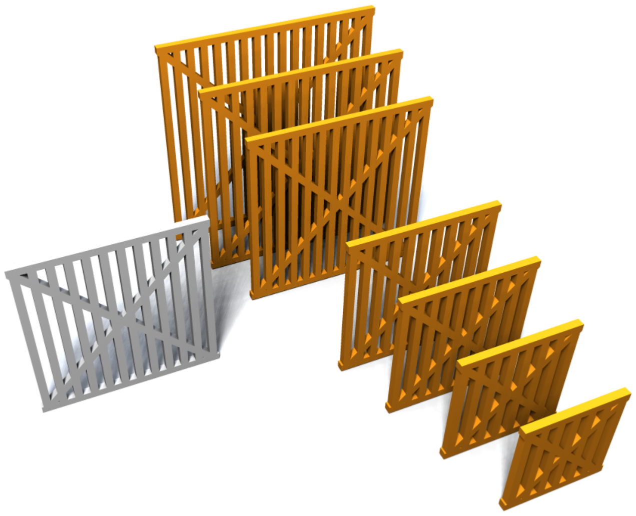


Figure 8: *A shape with overlapping patterns (grey). The approach of Bokeloh et al. [2011] cannot create discrete variations for such shapes because it requires a volumetric region with regularly repeating geometry. In this shape, the repeating geometry is broken by the diagonal elements. Our formulation handles such shapes natively and offers a degree of freedom for pattern-aware editing (orange).*

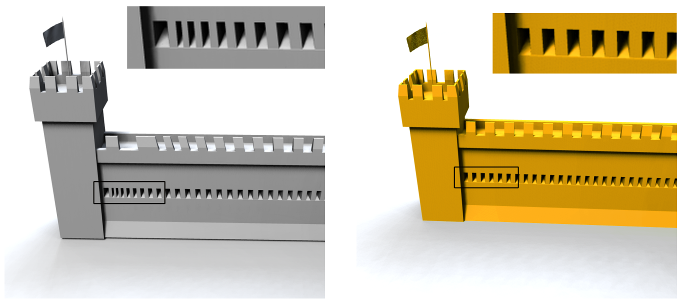
SVD implementation in terms of high input complexity and a large nullspace basis. Second, many shapes contain types of regularity that are not yet included in our model, such as rotations, complex hierarchical dependencies, and curved patterns. The first problem could be addressed by employing an alternative method for computing a basis for the null space. We have performed some preliminary experiments with factorizations optimized for sparsity [Gilbert and Health 1987], with promising results. The second issue is more fundamental, as we have to deal with non-linear objectives and, for general curved patterns, more general pattern detection. We leave this as a research challenge for future work.

7 Discussion

We have presented an algebraic regularity model for shape editing. The main idea is to detect regular patterns (and their continuous extensions) and describe the overall shape by linking their parameters. We derive a system of linear equations with linear inequality constraints that characterizes the space of valid variations of the shape. Interactive manipulation operations are modeled as quadratic objective terms. We demonstrate the algebraic model in a prototype shape editing system where the user can rapidly obtain plausible shape variations by placing and moving a set of sparse handles.

One could interpret our model as a formulation of Poisson surface editing in terms of degrees of freedom due to structural regularity in the input shape. The model maintains the appearance of man-made shapes more faithfully than previous work and naturally permits adaptation of the shape's structure while preserving its global characteristics.

We see our approach as a step towards structure models that describe spaces of related shapes using only basic and abstract assumptions, such as symmetry and correspondence. In future work, it would be interesting to study more complex classes of transformations, including rotations and scaling. One of the challenges here is the numerical treatment of non-convex motion trajectories. As



a) Bokeloh et al. [2011] b) our method

Figure 9: *Unlike the prior technique, our method does not create any residual bending artifacts.*

demonstrated by our results, imposing algebraic structure offers an improved understanding of structural properties of shapes and facilitates navigation in resulting shape spaces.

Acknowledgements

We thank Philipp Krähenbühl, Arjun Jain, Chuong Nguyen, and the anonymous reviewers for comments on this paper. This project was supported by the Cluster of Excellence "Multimodal Computing and Interaction" and the Max-Planck-Center for Visual Computing and Communication. This work was conducted in conjunction with the Intel Science and Technology Center for Visual Computing.

References

ALIAGA, D., ROSEN, P., AND BEKINS, D. 2007. Style grammars for interactive visualization of architecture. *IEEE Transactions on Visualization and Computer Graphics 13*, 4.

BOKELOH, M., WAND, M., AND SEIDEL, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Transactions on Graphics 29*, 4.

BOKELOH, M., WAND, M., KOLTUN, V., AND SEIDEL, H.-P. 2011. Pattern-aware shape deformation using sliding dockers. *ACM Transactions on Graphics 30*, 6.

BOTSCH, M., AND SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1.

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

DE BERG, M., CHEONG, O., VAN KREVELD, M., AND OVERMARS, M. 2008. *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer Verlag.

GAL, R., SORKINE, O., MITRA, N. J., AND COHEN-OR, D. 2009. iWires: an analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics 28*, 3.

GILBERT, J. R., AND HEALTH, M. T. 1987. Computing a sparse basis for the null space. *SIAM Journal on Algebraic and Discrete Methods 8*, 3.

KRAEVOY, V., SHEFFER, A., SHAMIR, A., AND COHEN-OR, D. 2008. Non-homogeneous resizing of complex models. *ACM Transactions on Graphics 27*, 5.

LIN, J., COHEN-OR, D., ZHANG, H., LIANG, C., SHARF, A., DEUSSEN, O., AND CHEN, B. 2011. Structure-preserving retargeting of irregular 3D architecture. *ACM Transactions on Graphics 30*, 6.

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

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

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proc. SIGGRAPH*, ACM.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proc. Symposium on Geometry Processing*, ACM.

TALTON, J. O., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis procedural modeling. *ACM Transactions on Graphics 30*, 2.

ŠT'AVA, O., BENEŠ, B., MĚCH, R., ALIAGA, D. G., AND KRIŠTOF, P. 2010. Inverse procedural modeling by automatic generation of L-systems. *Computer Graphics Forum 29*, 2.

XU, W., WANG, J., YIN, K., ZHOU, K., VAN DE PANNE, M., CHEN, F., AND GUO, B. 2009. Joint-aware manipulation of deformable models. *ACM Transactions on Graphics 28*, 3.

YANG, Y.-L., YANG, Y.-J., POTTMANN, H., AND MITRA, N. J. 2011. Shape space exploration of constrained meshes. *ACM Transactions on Graphics 30*, 6.