Analysis and Manipulation of Repetitive Structures of Varying Shape

Dissertation

zur
Erlangung des Doktorgrades (Dr.rer.nat)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Dipl. Inform. Alexander Berner
aus
Nürtingen

Bonn 2012

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms Universität Bonn

1. Gutachter: Prof. Dr. Reinhard Klein
2. Gutachter: Prof. Dr. Andreas Schilling

Tag der Promotion: 22.04.2013
Erscheinungsjahr: 2013

Summary

Self-similarity and repetitions are ubiquitous in man-made and natural objects. Such structural regularities often relate to form, function, aesthetics, and design considerations. Discovering structural redundancies along with their dominant variations from 3D geometry not only allows us to better understand the underlying objects, but is also beneficial for several geometry processing tasks including compact representation, shape completion, and intuitive shape manipulation.

To identify these repetitions, we present a novel detection algorithm based on analyzing a graph of surface features. We combine general feature detection schemes with a RANSAC-based randomized subgraph searching algorithm in order to reliably detect recurring patterns of locally unique structures. A subsequent segmentation step based on a simultaneous region growing is applied to verify that the actual data supports the patterns detected in the feature graphs. We introduce our graph based detection algorithm on the example of rigid repetitive structure detection. Then we extend the approach to allow more general deformations between the detected parts. We introduce subspace symmetries whereby we characterize similarity by requiring the set of repeating structures to form a low dimensional shape space. We discover these structures based on detecting linearly correlated correspondences among graphs of invariant features. The found symmetries along with the modeled variations are useful for a variety of applications including non-local and non-rigid denoising.

Employing subspace symmetries for shape editing, we introduce a morphable part model for smart shape manipulation. The input geometry is converted to an assembly of deformable parts with appropriate boundary conditions. Our *method uses self-similarities from a single model or corresponding parts of shape collections as training input and allows the user also to reassemble the identified parts in new configurations, thus exploiting both the discrete and continuous learned variations while ensuring appropriate boundary conditions across part boundaries.*

We obtain an interactive yet intuitive shape deformation framework producing realistic deformations on classes of objects that are difficult to edit using repetition-unaware deformation techniques.

Contents

1 Introduction

Self-similarity and repetitions are ubiquitous in man-made and natural objects. Such structural regularities often relate to form, function, aesthetics, and design considerations. Discovering structural redundancies along with their dominant variations from 3D geometry not only allows us to better understand the underlying objects, but is also beneficial for several geometry processing tasks including compact representation, symmetrization, shape completion, and intuitive shape manipulation.

Our goal is, given an input model like one in Figure 1.1, to automatically identify recurring parts and compute dense mapping functions between the parts. We approach this topic in increasingly challenging scenarios. First, we introduce a novel method for rigidly repeating shape detection. In this case the identified repetitive parts of the object have approximately the same shape but have been translated and rotated. An example for this class are the recurring windows and structures in Figure 1.1a. Then we extend this method to handle non-rigid repetitions, up to very large variations in the parts, e.g. the

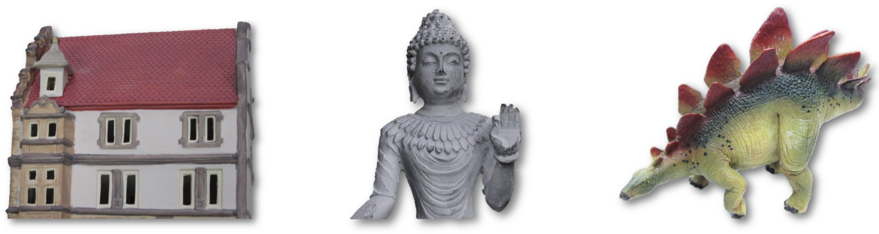


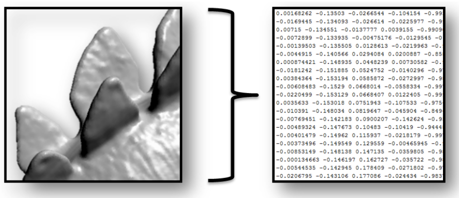
Figure 1.1: *Photos of input models used in this work: (a) Clay house model: approximately rigid repetitions (windows), (b) Buddha statue - non-rigid repetitions (ornaments), (c) Stegosaurus statue - intensive shape variations in recurring parts (back plates). In order to run our algorithms, we acquired these shapes using a Minolta laser triangulation scanner.*

ornaments of the statue and the back plates of the Stegosaurus in Figure 1.1b and c.

In the literature the definition of identifying recurring parts in one object is often referred to as "partial symmetry problem", as introduced by (Mitra et al. 2006). A stricter notion is to use the term "symmetry" only when algebraic regularity is involved (Mitra et al. 2012); we stick to the common convention of using the terms of recurring shapes in an object and partial symmetries interchangeably throughout this thesis.

Detection of partial symmetries in 3D objects received a lot of attention in the geometry processing literature (Mitra et al. 2006; Podolak et al. 2006; Loy and Eklundh 2006; Mitra et al. 2007; Pauly et al. 2008). An *EG State of the Art Report* (Mitra et al. 2012) presents an overview of the current techniques. Most of the previous work considers only reflections, rigid mappings and sometimes uniform scaling as mapping functions. However, many real-world objects show forms of structural redundancy that cannot be captured by such simple affine maps (Figure 1.1c). The different tail spikes and plates of the Stegosaurus input model might clearly appear similar to a human observer, but cannot be transformed into each other that way.

In our work, we develop an algorithm that is able to remove the restrictions of rigid symmetry detection, allowing useful applications that employ the gained structure information including non-local non-rigid denoising, model completion, simultaneous instance replacement and compression.



```
0.00168262 -0.13503 -0.0266544 -0.104154 -0.99
-0.0169445 -0.134093 -0.026614 -0.0225977 -0.9
0.00715 -0.134551 -0.0137777 0.0039155 -0.9909
-0.0072899 -0.133935 -0.00475176 -0.0129545 -0
-0.00139503 -0.135505 0.0128613 -0.0219963 -0.
-0.0044915 -0.140566 0.0294084 0.0200887 -0.85
0.000874421 -0.148935 0.0448239 0.00730582 -0.
-0.0181242 -0.151885 0.0524752 -0.0140296 -0.9
0.00384364 -0.153194 0.0585872 -0.0272997 -0.9
-0.00608483 -0.1529 0.0668014 -0.0558334 -0.99
-0.0220499 -0.153129 0.0668407 0.0122405 -0.99
0.0035633 -0.153018 0.0751943 -0.107533 -0.975
-0.010391 -0.148034 0.0819647 -0.045904 -0.849
-0.00769451 -0.142183 0.0900207 -0.142624 -0.9
-0.00489324 -0.147673 0.10483 -0.10419 -0.9444
-0.00401479 -0.14962 0.115937 -0.0218179 -0.99
-0.00373496 -0.149549 0.129559 -0.00465945 -0.
-0.00853149 -0.148138 0.147135 -0.0359805 -0.9
-0.000134663 -0.146197 0.162727 -0.035722 -0.9
-0.00544535 -0.142945 0.178409 -0.0271802 -0.9
-0.0206795 -0.143106 0.177086 -0.024434 -0.983
```

Figure 1.2: *Representation of shape data: As humans, we immediately see structure in the back plates of the scanned Stegosaurus model. But for the computer, this is just an unstructured list of numbers (xyz-positions). Our motivation is to find algorithms that identify the same structures in this data set as humans are able to by looking at the shape.*

Further, we introduce a novel symmetry-based shape deformation technique, showing superior results over non-symmetry aware methods, that is learning allowable deformations from shape varying repetitions.

We will now formalize our notation of recurring parts and the computed dense mapping functions.

1.1 Problem Statement

We address the problem of generalized partial symmetry detection (Figure 1.3). Given an input object $\mathcal{S}$, our goal is to identify a piece of geometry $\mathcal{U} \subset \mathcal{S}$ and a collection of associated mapping functions $\mathbf{f}_i : \mathcal{U} \rightarrow \mathbb{R}^3$ that respectively create instantiations $\mathbf{f}_i(\mathcal{U})$, thereby matching the original geometry $\mathcal{S}$ approximately. The pieces of $\mathcal{S}$ covered by $\mathbf{f}_i(\mathcal{U})$ are called the parts $\mathcal{P}_i$. We call the (randomly chosen among all instances) part $\mathcal{U}$, where our set of mappings is defined, the *urshape*.

We define $\mathcal{F}$ as the family of transformations allowed for $\mathbf{f}_i$ in the search for repetitions. For example, we can only allow rigid mappings (translation, rotation and mirroring). In this case, $\mathbf{f}_i$ can be written as one 4×4 Matrix $\mathbf{T_i}$ to map all points $\in \mathcal{U}$.

During this thesis, we are aiming to allow more general transforms for f_i. We approach this in three steps: First we introduce a novel, feature graph-based symmetry detection method for rigid symmetry detection, where f_i is actually restricted to rigid mappings $\mathbf{T_i}$. But the advantage of our method in

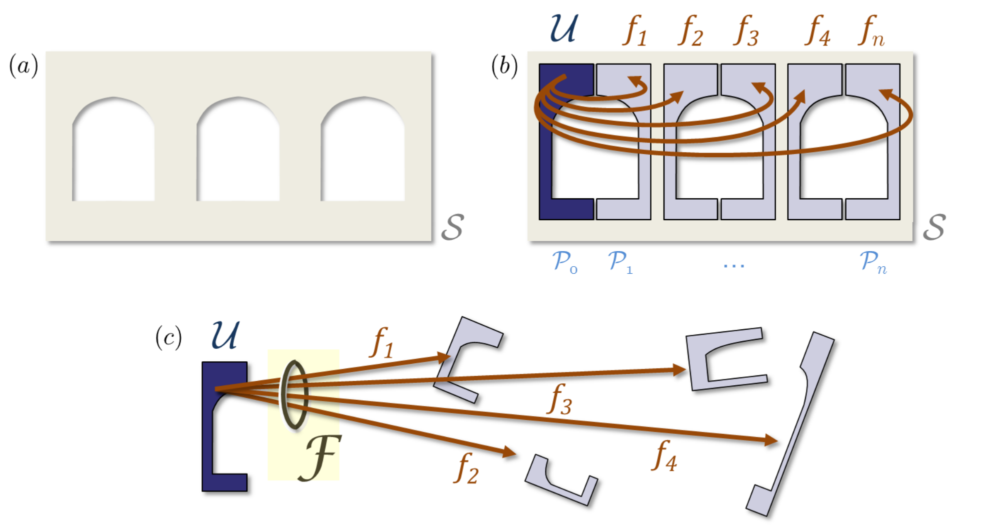


Figure 1.3: *(a) Input shape $\mathcal{S}$, (b) detected part $\mathcal{U}$ with dense correspondence mappings f_i to other detected instances of this part $\mathcal{P}_i$, (c) starting from rigid mappings as shown in (b), we extend the families of mappings $\mathcal{F}$, to allow nonrigid deformations in the search for repetitions. The repetitive structures can be positioned anywhere on $\mathcal{S}$. We do not require a regular grid or similar regular arrangements of the instances.*

comparison to other approaches is that our algorithm is naturally extendable to non-rigid symmetry detection.

We show how to conduct this in the second step, where we extend the family of mappings to approximately isometric deformations controlling the sensitivity to non-isometry by relaxed tolerance parameters.

Allowing more general mappings f_i between the repetitive parts on the surface uses a larger number of mapping parameters. This can lead to overfitting and spurious matches, while an overly restrictive mapping fails to compactly capture redundancy present in the input. To solve this, we introduce subspace symmetries in the third step, whereby we characterize similarity by requiring the set of symmetric parts to form a low dimensional shape space.

1.2 Contributions

Our contributions can be split into three main topics (Figure 1.4): In the first part of this work, we introduce the approach by solving the partial approximate rigid symmetry detection problem based on building graphs of salient features and an efficient subgraph matching technique. In the design of our algorithm we focus on the detection of symmetries in scanned point cloud data sets, where our method handles real-world raw 3D scanner point clouds with noise artifacts robustly. In addition, we also consider generalizations to image data and triangle meshes, demonstrating the versatility of the novel approach.

We extend our method to detect general isometric symmetries in 3D shapes. Our algorithm can handle partial isometric symmetries as well as more general symmetries where the preservation of intrinsic geometric quantities can be relaxed.

Concluding deformable symmetry detection, we introduce a subspace symmetry model that removes the restrictions to simple, parameterized mapping functions. We propose a practical detection algorithm for models where subspace symmetries preserve distinct crease line features.

We evaluate these techniques on various data sets and show that for models with pronounced surface features, many repetitions can be found fully automatically. In all the shown stages, our technique computes dense correspondences and we subsequently utilize them in various applications, such as model repair, super-resolution, denoising and shape manipulation.

Employing the subspace symmetries paradigm for a novel symmetry-based editing technique, we characterize the continuous allowable variations both for the individual parts and their interconnections. We obtain an interactive yet intuitive shape deformation framework producing realistic deformations on classes of objects that are difficult to edit using structure un-aware deformation techniques. We explore applications such as partial symmetrization, caricatures and present first steps towards generalized inverse procedural modeling.

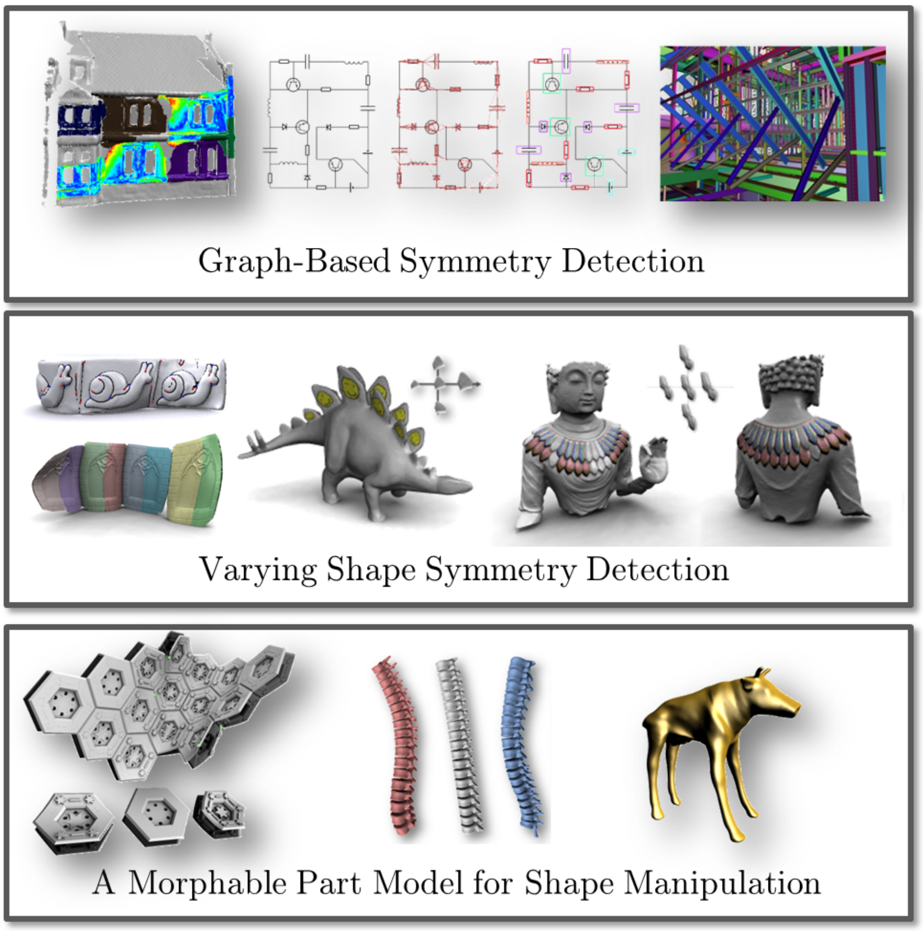

Graph-Based Symmetry Detection
Varying Shape Symmetry Detection
A Morphable Part Model for Shape Manipulation

Figure 1.4: *Our contributions can be split into three main topics: We introduce a novel approach for the partial rigid symmetry detection problem employing subgraph matching. We propose non-rigid partial symmetry detection algorithms and show how to control the larger degrees of freedom. And we utilize found dense part correspondences in a symmetry-based intuitive shape manipulation method.*

1.3 List of Publications

The following list contains all the work published during my PhD studies. The publications are separated into two groups. This dissertation is based on publications of the first list. The second list represents another scope of work that has been done during the PhD study.

The main text passages of the papers from the first list have been utilized in this thesis without being tagged individually. The corresponding coauthors have permitted the use of these text passages as well as figures from the originally published papers.

Berner, A. ; Bokeloh, M. ; Wand, M. ; Schilling, A. ; Seidel, H.-P.: A Graph-Based Approach to Symmetry Detection. *In: IEEE/EG International Symposium on Volume and Point-Based Graphics. Los Angeles, CA : Eurographics Association, 2008*

Berner, A.; Bokeloh, M.; Wand, M.; Schilling, A. ; Seidel, H.-P.: Generalized intrinsic symmetry detection *In: Max-Planck-Institut Informatik, Tech Report MPI-I-2009-4-005, August 2009*

Bokeloh, M. ; Berner, A. ; Wand, M. ; Seidel, H.-P. ; Schilling, A.: Symmetry Detection Using Line Features. *In: Computer Graphics Forum (Proc. Eurographics 2009) 28 (2009), Nr. 2*

Berner, A. ; Wand, M. ; Mitra, N. ; Mewes, D. ; Seidel, H.-P.: Shape Analysis with Subspace Symmetries. *In: Computer Graphics Forum (Proc. Eurographics), 2011*

Berner, A.; Burghard, O.; Wand, M.; Mitra, N. J.; Klein, R. ; Seidel, H.-P. A Morphable Part Model for Shape Manipulation *In: Max-Planck-Institut Informatik, Tech Report MPI-I-2011-4-005, December 2011*

1.3.1 Additional Publications

Wand, M.; Berner, A.; Bokeloh, M.; Fleck, A.; Hoffmann, M.; Jenke, P.; Maier, B.; Staneker, D. Schilling, Interactive Editing of Large Point Clouds *In: Symposium on Point-Based Graphics 2007 : Eurographics / IEEE VGTC Symposium Proceedings, Eurographics Association,* *2007*

Wand, M.; Berner, A.; Bokeloh, M.; Jenke, P.; Fleck, A.; Hoffmann, M.; Maier, B.; Staneker, D.; Schilling, A. Seidel, H.-P. Processing and interactive editing of huge point clouds from 3D scanners *In: Comput. Graph., Pergamon Press, Inc., 2008*

Bokeloh, M. ; Berner, A. ; Wand, M. ; Seidel, H.-P. ; Schilling, A.: Slippage Features *In: Technical Report, WSI-2008-03, Wilhelm Schickard Institut, University of Tuebingen. 2008*

Wand, M.; Adams, B.; Ovsjanikov, M.; Berner, A.; Bokeloh, M.; Jenke, P.; Guibas, L.; Seidel, H.-P. Schilling, A. Efficient reconstruction of nonrigid shape and motion from real-time 3D scanner data *In: ACM Transactions on Graphics (TOG) Volume 28(2), April 2009*

Tevs, A.; Berner, A.; Wand, M.; Ihrke, I. Seidel, H.-P. Intrinsic Shape Matching by Planned Landmark Sampling *In: Computer Graphics Forum (Proc. EUROGRAPHICS), Blackwell, 2011, 543-552*

Tevs, A.; Berner, A.; Wand, M.; Ihrke, I.; Bokeloh, M.; Kerber, J. Seidel, H.-P. Animation Cartography - Intrinsic Reconstruction of Shape and Motion *In: ACM Transactions on Graphics, 31(2), April 2012 (presented at SIGGRAPH2012, Los Angeles, CA)*

1.4 Outline of the Thesis

In Chapter 2, we introduce our graph-based approach starting with rigid symmetry detection. We extend this method with a relaxed approximate isometric symmetry detection method in Chapter 3. Using many more degrees of freedom for symmetry detection, we propose a stable subspace model in Chapter 4. Finally in Chapter 5, we apply the subspace model in shape deformation and content creation. Details on related work are given in the respective chapters.

2 Graph-Based Symmetry Detection

Throughout this thesis, we develop a method that is able to identify shape varying repeated parts of an input model. In this chapter, we introduce our graph based approach for the symmetry detection problem firstly starting with rigid transformations. In the next chapters, we will extend this approach to general mapping functions between symmetric parts. In the rigid problem setting, we designed our method focusing on input geometry represented as point clouds.

In this chapter, we restrict the family of allowed mapping functions $\mathcal{F}$ between the parts to rigid transformations $\mathbf{T}_i$ (translations and rotations). Many real-world objects consist of parts that are approximately rigid similar to each other. For example, a facade of a building contains a number of windows with comparable geometry.

2.1 Related Work

Partial rigid symmetry detection in 3D shapes has gained a lot of interest in literature. The probably most successful techniques so far are based on transformation voting: A set of candidate correspondences between surface points is estimated and a transformation between the according local neighborhoods is established. Similar to a Hough transform, a voting procedure in transformation space is used to identify well-supported transformations. Having extracted such clusters of transformations, the according correspondences between surface patches can be easily determined. Mitra et al. (2006) propose this type of algorithm, using principal curvature directions to create votes in a transformation space of rotation, translation and scaling. A mean-shift algorithm is used to extract dominant transformation clusters. The technique can be extended to an optimization technique that makes approximately symmetric objects more symmetric (Mitra et al. 2007). Another voting approach has been proposed concurrently by (Podolak et al. 2006), who use voting on reflective planes to detect planar reflective symmetries. Gal and Cohen-Or (2006) form clusters of quadratic surface patches and use geometric hashing (Lamdan and Wolfson 1988) to find symmetries in objects. In image data, a Hough transform of salient feature points has been used by Loy and Eklundh (2006) to find symmetric configurations. Martinet et al. (2006) propose a technique that uses a transformation to generalized moment functions in order to compute global symmetries of 3D shapes. Localized symmetries are detected by applying the algorithm hierarchically to detected parts. A related strategy is proposed by (Simari et al. 2006): Planar reflective symmetries are detected by computing an auto-alignment of parts of a shape with itself. The iterative alignment process is initialized by a PCA-based split of the object in halves; afterwards, an iteratively reweighted least-squares alignment is computed, where the reweighting cuts off outliers that correspond to non-symmetric parts of the object. Hubo et al. (2007) detect symmetries based on local descriptors and perform a compression by aligning matching heightfields and performing a PCA analysis of the resulting space of example shapes. Kazhdan et al. (2003) analyze objects for central symmetry and use this as a descriptor for shape retrieval. Another very interesting application of symmetry detection is shape completion: Thrun and Wegbreit (2005) compute symmetries of partially scanned objects using a brute force search and local descent algorithm and use this information to complement the partially acquired shape, taking occlusion constraints of the acquisition process into account.

In contrast to methods based on voting (Mitra et al. 2006; Gal and Cohen-Or 2006; Podolak et al. 2006; Loy and Eklundh 2006), our algorithm is based on a graph matching strategy. The main advantage of this approach is that it can be generalized to more general matching criteria. With voting methods, the dimensionality of the transformation space increases with additional degrees of freedom which makes the detection problem harder to solve and computationally less efficient. In addition, our algorithm can handle both local and global symmetries, the scale being only determined by the level of detail that is used in the feature detector. In contrast, global methods (Kazhdan et al. 2003; Martinet et al. 2006; Simari et al. 2006) have to detect symmetries in a top-down fashion, relying on an initial symmetric decomposition. Voting methods are also affected by this problem, as all votes are cast into the same transformation space. Without some kind of partition, the transformation space might become cluttered so that detailed symmetries are hard to detect. Graph-based pattern matching techniques have been explored in computer vision: Felzenszwalb and Huttenlocher (2005) use tree-shaped graphs of object parts with an appearance model to infer deformable shape configurations in images. A graph-based algorithm for 3D object retrieval has been proposed by Schnabel et al. (Schnabel et al. 2008): First, shape primitives are fitted to a point cloud, then the structure of the incidence graph is learned and used to retrieve the object within a larger collection using a subgraph matching algorithm.

2.2 Overview

The pipeline of our symmetry detection algorithm is outlined in Figure 1: We start by detecting locally unique features on the geometry. For this step, we use the slippage feature algorithm (Bokeloh et al. 2008), which is capable of

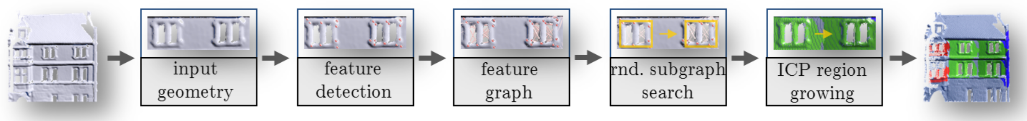


Figure 2.1: ***Processing pipeline:*** *First a set of locally unique features is detected, which then form a graph on the object surface. Next, subgraphs with matching graph connectivity and approximately matching geometric embedding are extracted. From these discrete candidate matches, data points are assigned to symmetric regions using an ICP-based simultaneous region growing algorithm, which yields the final result.*

detecting features in very general settings. From these features, we build a neighborhood graph that describes the coarse scale similarity structure of the object. Details on this step are given in Section 2.3. Given this graph, we employ a randomized subgraph search algorithm in order to detect recurring patterns in this graph (Section 2.4). Mapping the search for similarities to a subgraph matching problem reduces the amount of information that needs to be processed dramatically, which allows for an efficient solution to this problem. In order to make sure that the reduced, discretized solution matches the continuously defined geometry, we perform a final validation using a variant of an iterative-closest-points (ICP) registration algorithm (Besl and Mckay 1992; Chen and Medioni 1992) that performs simultaneous matching and region growing over all detected patterns (Section 2.5). Generalizations of this basic strategy are discussed in Section 2.7. Finally, we evaluate our algorithm on various test data sets (Section 2.8) of rigid symmetries.

2.3 Building the Feature Graph

2.3.1 Feature Detection

Our approach is based on matching graphs of arbitrary feature points. However, the chosen keypoint detector is decisive for the attainable quality of the results. A big problem with feature detection techniques is that they typically restrict themselves to a certain class of geometric features (such as bumps, i.e. points on the surface where both principal curvatures are large). Such a restriction is meant to make the detection more reliable; however, it strongly restricts the class of feature points that can be detected. As a consequence, many regularities in objects might remain unnoticed by a feature-based symmetry detector, as no features might be available in many regions of the object.

We deal with this problem by choosing the "slippage features" detection algorithm of (Bokeloh et al. 2008) that can detect keypoints of arbitrary type with the same reliability as state of the art techniques, e.g. (Li and Guskov 2005). *As in our work feature detection methods are not the focus, we give only a brief overview of the chosen detector; for further details and a quantitative evaluation, see (Bokeloh et al. 2008).*

The "slippage feature" detector works directly on point clouds. For each input data point p_i, it analyses a spherical neighborhood. The key idea is: in order for p_i to be a good feature point, the auto-alignment problem of it with itself should be uniquely defined. This means, if we register this piece of geometry with itself, a unique position and orientation should result. A necessary requirement for this property is that the "slippage" of the piece of geometry is small. (Gelfand and Guibas 2004) set up an ICP alignment objective function that measures the squared point-to-plane distance for this patch with itself at the given location. Obviously, the error itself will be zero but the Hessian matrix of the error function with respect to rotations and translation will reveal how well conditioned the auto-alignment problem is. The keypoint detector computes a slippage value for all surface points and performs mean shift clustering in order to extract local extrema of this measure. These extrema become the final feature points. To deal with features at various scale levels, this algorithm is extended by performing the extraction for several neighborhood sizes $\epsilon_1, , \epsilon_k$, doubling in each step: $\epsilon_{i+1} = 2\epsilon_i$. The correct scale for the features is determined automatically by running mean shift clustering on the 3-dimensional manifold of surfaces in scale space.

We denote the resulting keypoint feature points as $k_i, i = 1...n$. The scale is given by $\epsilon(k_i)$ and the associated neighborhood of input points inside a radius $\epsilon(k_i)$ by $N(k_i)$.

Comparing individual features points: In order to detect matching keypoints, we further follow (Bokeloh et al. 2008) employing a descriptor based on curvature histograms in concentric rings within each $N(k_i)$. Comparing these histograms is fast and results in a matching score that we use later when we compare possible subgraphs of keypoints.

2.3.2 Graph Generation

Given the computed feature points, we construct a graph that connects feature points with each other in a local neighborhood. From a theoretical point of view, a complete graph of all connections would provide the richest pool of subgraphs to examine for recurring patterns. However, the solution to the matching problem in such a densely connected graph becomes too expensive in practice. Therefore, we build only a k-nearest neighbor graph of features (typically: $k = 20$). The underlying assumption for this simplification is that first, recurring patterns related to symmetries we want to detect are locally coherent: If features far away correlate with each other, there will be other features in between that belong to the same symmetry. Second, we assume that we might miss a few feature points that drop out of our detection scheme but it is unlikely to miss a large number (such as 20 neighbors) at the same time. Therefore, the restriction to a k-nearest features graph is sufficient for our problem setting. We denote the feature graph by $G = (K, E), K = \{k_1, \cdots, k_n\}$ and edges E.

2.4 Sub-Graph Matching

2.4.1 Problem Statement

Given a graph of features, we want to determine corresponding subgraphs. This means, we want to identify disjoint subsets $S_j^{(i)} \subseteq K$ that are symmetric. The index i refers to the class of symmetric subgraphs, ranging from 1 to n_S, and j to the instance index within each class: All *instance sets* $S^{(i)} := \{S_1^{(i)}, ..., S_{\#S^{(i)}}^{(i)}\}$ describe parts of the graph that are similar to each other. We refer to the whole collection of instance sets as a *symmetry set* S. Similarity within an instance means that the corresponding subgraphs are close to each other according to a distance function $dist_G$:

$$\forall i \in \{1 \cdots n_s\} : \forall j_1, j_2 \in \{1 \cdots \#S^{(i)}\} : dist_G(S^{(i)}_{j_1}, S^{(i)}_{j_2}) \leq \epsilon$$

A set of subgraphs where all pairs of elements are similar to each other form an instance set $S^{(i)}$. We allow for multiple instance sets, describing different classes of similarities (such as windows, doors, ...), but these instances have to consist of disjoint features k_i. Please note that we use a distance function to define subgraph similarity, not an equivalence relation. In particular, our similarity is not transitive; it is possible that subgraphs g_1 and g_2 are similar, as well as g_2 and g_3, but not g_1 and g_3, because the distance between these two is larger than the permitted threshold. Therefore, we demand pairwise similarity of all subgraphs within each instance set. We might also compute different maximal instance sets that contain common subgraphs but which are not the same. Thus, in practice, the result might depend on the subgraph at which the search has been initiated. This is an inherent problem; in general, a strict equivalence relation cannot be defined without making arbitrary decisions (think of two shapes morphing into each other). We address this problem by resorting to a pairwise matching criterion and use a Random Sample Consensus (RANSAC) algorithm (Fischler and Bolles 1987) to maximize the descriptive power of the extracted instances.

2.4.2 The Subgraph Distance Function

As distance function, we concentrate on a simple rigid matching function in this chapter. This is the first step in exploring graph-based symmetry detection. It will be conceptually straightforward to apply more general matching criteria such as a graph matching with isometric rather than Euclidean embedding as shown in Chapter 3.

The rigid matching criterion needs correspondences to be established between feature points in all subgraphs of an instance. This will be done automatically during the graph matching algorithm (see Section 2.4.5). From these correspondences, a least-square optimal transformation matrix is computed that maps corresponding points to each other. We then form a score based on the average distance of feature points to their transformed corresponding points, the matching of their descriptors and average distortion of length of edges in the graph. Two subgraphs are similar, if the rigid mapping leads to a score below a user defined threshold ϵ.

2.4.3 Graph Matching Objectives

Our symmetry detection criterion is capable of detecting a large variety of valid symmetries $S = \{S^{(1)}, \dots, S^{(N)}\}$. Therefore, we proceed in two steps: First, we want to compute maximal symmetries. However, usually even a large number of such maximal solutions exist. Therefore, in a second step, we look for well-supported solutions; we regard a solution as more convincing if a large number of corresponding features and instances give evidence that it is not spurious.

Maximal symmetries: A symmetry set S is maximal, if no more similar subgraph can be added to the solution without violating the previously defined conditions. Because of the disjointedness requirement, we can optimize an existing symmetry set in multiple, possibly competing, directions: On the one hand, we can add additional subgraphs to an instance set, which means the same pattern has been found at more places than previously. We refer to such an operation as *instance expansion*. A second operation is adding more features to an instance. This means, we retain the same number of patterns that are similar, but make each subgraph larger by adding another feature that is recurring in all subgraphs of an instance. We call such an operation a *feature expansion*. A third operation is adding a new instance set to the symmetry set. This means, we detect a new pattern, not matching previously known ones that appears multiple times in our feature graph G. We call such an operation a *pattern expansion*. As all subgraphs in a symmetry set are forced to be disjoint, all of these moves are competing, leading to different, mutually exclusive ways of creating maximal symmetry sets. It is easy to see that enumerating all valid, maximal possibilities might lead to an exponential number of solutions.

In the following, we will develop an expansion strategy that creates maximal sets according to heuristic regularity rules that yield ”reasonable” maximal sets. By restarting the search algorithm on different random initializations following the RANSAC paradigm, we compute an estimate of an optimal solution in the sense of being the best supported of all regular, maximal solutions.

2.4.4 Inherent Ambiguities

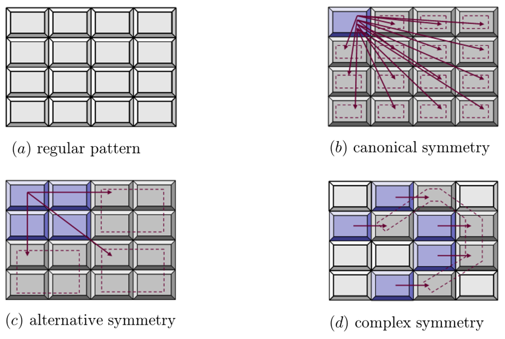


Figure 2.2: *A simple regular pattern can exhibit numerous maximal symmetries.*

A fundamental problem in symmetry detection is the phenomenon of inherent ambiguities. This can be understood by looking at typical regular patterns that occur in particular in geometries of man-made objects. For example, consider a wall consisting of an array of bricks, laid out on a simple regular grid (Figure 2.2). We could consider instances of bricks, covering the whole wall. Alternatively, we could also form instance sets consisting of subsets of bricks, such as sets of adjacent bricks or even irregular subsets that recur one or more times. Such regular structures can be described using group theory (Mitra et al. 2006): We consider the group of rigid transformations. The transformations T that map regular patterns to a symmetric configuration such as depicted in Figure 2.2 can be described by repeated application of generator transformations G_i:

$$\mathbf{T}_{i_1,\dots,i_k} = G_1^{i_1} \circ G_2^{i_2} \circ \cdots \circ G_k^{i_k}$$

Where the range of exponents i_j is subject to additional index constraints (in our example, the instance is only valid for $i_1, i_2 \in \{0, \ldots, 3\}$ if $\mathbf{T}_1$ moves a brick to the left, and $\mathbf{T}_2$ downwards by one). Given a product that satisfies the index constraints, any multiplication with products of generators will again form a valid instance if the overall product still satisfies the constraints.

In order to get reasonable results, we need to resolve these ambiguities. For our algorithm, we have decided to impose an additional regularity constraint to make the algorithm find the simplest and most frequently occurring instantiation patterns. This will yield the smallest possible subgraphs that are instantiated as much as possible. In terms of the group of rigid motion, it will try to find solutions with transformations that have no more common divisor.

2.4.5 Matching Algorithm

With these considerations at hand, we can design a subgraph matching algorithm. Our algorithm is based on a RANSAC approach: We start with small seed symmetry sets and extend these to maximal sets using the previously described expansion moves.

Initialization: The outer loop of the RANSAC graph matching algorithm creates the smallest possible non-trivial subgraphs, which are edges from the graph, with candidate correspondences. For this, we compare all edges to all other edges according to our subgraph distance measure $dist_G$. This means, we select all pairs (e_i, e_j) of edges where the features at both end points match between the two edges and the edge length is within the defined threshold ϵ. We assign an importance value of $(\epsilon - dist(e_i, e_j))$ to each edge correspondence and add up the score for all recurrences of each edge, thus favoring edges that recur more frequently. We then use importance sampling according to this importance value to randomly create an initial symmetry set that contains only one instance of one single edge recurring several times in our model.

Expansion: In order to compute the most regular, i.e. the simplest possible, building blocks for our symmetry set, we chose to always first perform instance expansion to maximize our current candidate symmetry set, followed by feature expansion. This means, we first find all edge correspondences to the selected initial edge. Afterwards, we add more features to these initial subgraphs until no more expansion is possible. This second step considers all edges incident to nodes in the current subgraphs. If length and angles of these outgoing edges match our error threshold in all subgraphs of the instance, the outgoing edge is added to the model. This step is repeated until no more expansion is possible. Once an instance is completed, additional, disjoint instances are computed by iterating this random sampling algorithm until no more solutions are found (pattern expansion).

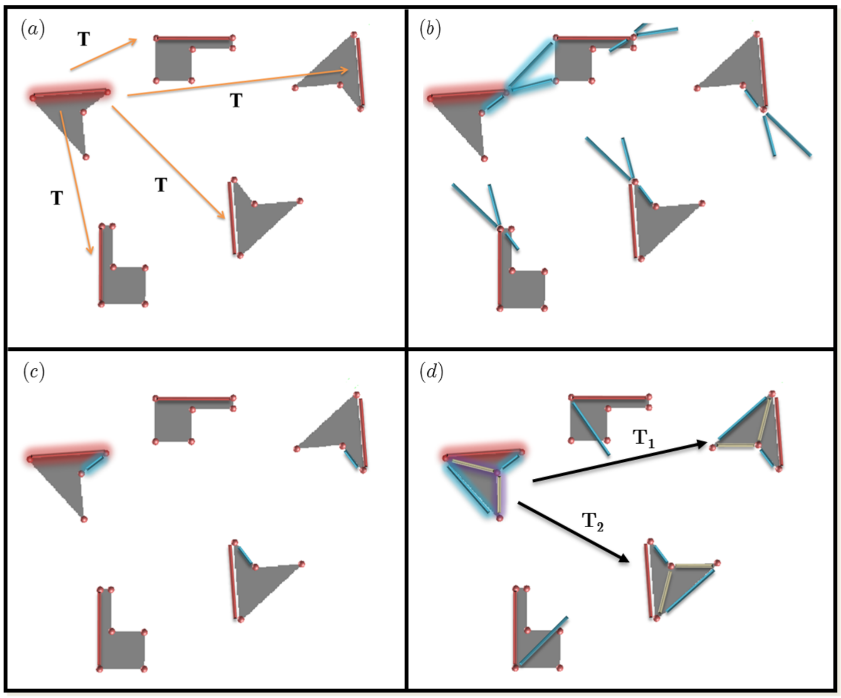


Figure 2.3: ***Matching algorithm basic principle:*** *Input is an image with corner features (red dots), for clarity reasons, the connecting k-nearest neighbor graph between the feature points is not shown in the images. We display the currently examined edges only. (a) We randomly pick a start edge (red) and compute a transformation to all matching edges with similar properties. (b) Using these transformations, we test adjacent edges (blue) and reject incorrect ones, as shown in (c). (d) After testing all adjacent edges, we discover three subgraph instances and the rigid transformations T_i.*

Outer loop: The outer loop of the RANSAC algorithm evaluates the symmetry sets obtained this way. We perform a number of iterations (typically 10 are sufficient) of the instance search, and then compute a score for each solution. The score is given by the number of instances, multiplied by the number of subgraphs in each instance, which again is weighted by the number of features involved in each of these. In this way, we compute the most complex solution, where symmetries are best supported in the sense of supported by the largest number of features and recurring subgraphs found.

Robustness to noise: Our method relies on distinguishable descriptors. If a data set is very noisy and incomplete, this is not the case. To avoid incorrect symmetries we add an ICP verification step in the pattern expansion: When we have decided to take an edge, we use the transformation matrix to copy a small piece of geometry to the target edge. Only if ICP converges there, the edge is taken. Too much noise also causes the algorithm to detect multiple classes of the same geometry due to slight variations in feature coverage, but it still retrieves symmetries for most objects. To solve this, we can apply a second stage after a class is completed: We take random edges of the found class members and search for them in the remaining data. This step reveals all instances in the noisy data experiments (Figure 2.7).

2.5 ICP-based Region Growing

Having found symmetric features constellations as described in the previous section, we now want to transfer these graph symmetries to the point cloud. We need this step to associate actual geometry with the discrete regularity patterns we have computed so far. First, we consider the case of a single instance set: This means, we are given symmetric subgraphs with sets of features $F_1, \ldots, F_m$, and rigid transformations $\mathbf{T}_{i,j}$ that map every feature point in F_i to the corresponding feature point in F_j (as computed by the graph matching algorithm with the rigid distance measure). We initialize region growing by putting every feature point $k \in F_1$ in a priority queue sorted by distance to the feature. We then iteratively pop the priority queue and add neighboring points to the queue if they fit the surface around F_1 when being transformed by $\mathbf{T}_{i,1}$ (neighborhood is determined by a precomputed $k-$nearest neighbor graph of the data point cloud; typically $k = 12$). At this point, we need a measure that defines the distance from an arbitrary point to the surface. Due to noise in the data and differences in discretization we cannot use the trivial nearest neighbor approach. Instead we use a variant of an MLS projection to define the surface and to measure the distance between the projected points. To project a point x, we compute a local tangent coordinate system via principal component analysis (PCA), fit a bivariate quadratic polynomial to the data points in the least squares sense and move the point x onto the computed polynomial. We use a standard Gaussian window function $\omega = \exp(\|x\|^2/\sigma^2)$ to define the support for PCA and least-squares fitting. We adapt the parameter σ to the amount of noise and variation in sampling density.

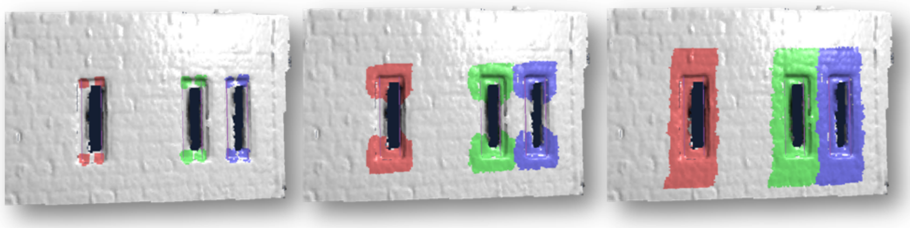


Figure 2.4: *Point-wise ICP-based region growing of three instances.*

Using this surface representation, we also compute a normal for the projected point. In order to decide whether points x and y from two potentially symmetric pieces of geometry match, we project both points onto their local surface, then transform the result in one coordinate system using $\mathbf{T}_{i,1}$, and measure the distance of the points and of the corresponding normals. If both differences are within a threshold, we have found a symmetric point on the geometry. For multiple instances, the test is executed for all pairs of points and considered successful if all pairs succeed. While we keep adding points to a region this way, we mark every visited point with an id according to the feature where the region growing has been started. If a point has been marked with another region id, we skip it so that we compute disjoint results, as in the discrete case. As we start from a discrete solution that maximizes the number of subgraphs in its instances first, and numbers of features involved second, we expect to perform region growing on the simplest, most elementary building blocks. Disjoint growing according to smallest distance now again tries to compute the simplest decomposition into building blocks. As demonstrated in the result section, this heuristic is not perfect, but yields reasonable results in practice. In order to handle multiple instance sets $S^{(i)}$, we perform the same algorithm simultaneously on all instances. In particular, points are associated with at most one instance so that the resulting symmetry set for the points is still strictly disjoint.

Improving the accuracy: The initial transformations $\mathbf{T}_{i,j}$ are not necessarily the best matching transformations between two symmetric patches. Especially smaller patches tend to have small errors in rotation. When a sufficient number of neighboring points are added to the region, we use ICP alignment to improve the transformations.

2.6 Grid-based Region Growing

The basic region growing algorithm with its test of every single point is too expensive for huge models. In (Bokeloh et al. 2009), we introduced an improved version of this method that is looking at surface pieces at once rather than isolated points. We impose a regular grid onto the urshape and treat all points within one grid cell simultaneously. In particular, the decision to include or not include a piece of geometry is now made per voxel cell, rather than per point. When the basic algorithm compares a piece of geometry, we set the radius r_{comp} to $2\times$ the grid cell diagonals. Please note that the voxel grid is imposed on the urshape only and not on the whole model; the geometry in the voxel is transformed to the instances for comparison. This avoids aliasing problems in the that would occur if we were comparing pairs of voxels.

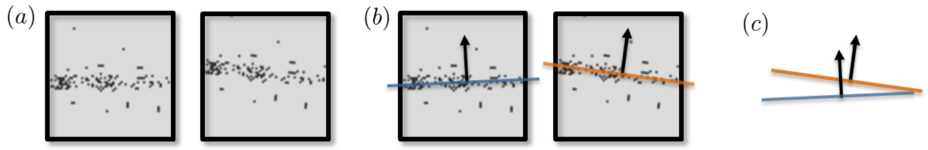


Figure 2.5: *Grid-based Region Growing: (a) We impose a regular grid onto the urshape and treat all points within one grid cell simultaneously. (b) We fit a plane to the data points in each instance and (c) compare the distance and the normal deviation of the planes to decide to grow over all points inside this grid cell.*

For the test of geometric mismatch, we cut out a small sphere $S^{(i)}_{comp}$ of fixed radius r_{comp} and compare the resulting geometry in each instance. Because a test of normal differences and position of a single point is very unstable for a noisy and irregular sampled point set we fit in each instance a plane to the data points in $S^{(i)}_{comp}$. We then compare the distance to the center of the sphere and the normal deviation between all pairs of instances (we use a threshold of 25°). If the geometry matches in most instances (we allow 20% of the checked instances to be outliers in order to make the algorithm more robust against structured noise, which is present in all our example data sets), the point is tagged as occupied by this instance, transformed back into the urshape by $(T)^{-1}_i$ and added to the urshape. Points that have already been occupied are not added to the urshape. In the other case, we add all neighbors within $S^{(i)}_{comp}$ to the priority queue to continue growing.

Handling holes: One application of symmetry detection in 3D scanner data is filling up acquisition holes and equalizing sampling density. Therefore, we need to be robust to acquisition holes. Our solution is to check the number of points within the spheres $S^{(i)}_{comp}$. If too few points are found, no reliable plane fit is possible and we treat the voxel as a “hole”. We use a relaxed threshold for outlier mismatches due to holes, allowing for matching partial data more robustly.

2.7 Extensions

Our symmetry detection scheme can be easily extended to other data modalities than point clouds. As an example, we apply the algorithm to bitmap images and triangle meshes. In the latter case, we assume that we have a consistently triangulated mesh that has perfect symmetries up to numerical precision. Detecting symmetric parts is useful in reverse engineering applications, where only a triangle soup of some original construction plan is known and the original instancing scheme has been lost and should be recomputed.

Images: In the case of symmetry detection in images, we employ the standard *OpenCV* corner detector (Harris and Stephens 1988) in order to compute feature points. As feature descriptor, we compute a simple histogram of color values within a circular neighborhood of fixed size (radius: 6 pixels). We then run the same graph matching algorithm as in the point cloud case (we actually use the same code, with feature representation encapsulated accordingly). Region growing is performed similar to the point cloud case; however, MLS projections are not necessary and thus omitted. In order to define the neighborhood of the "data points", we just employ the 4-neighborhood on the pixel grid.

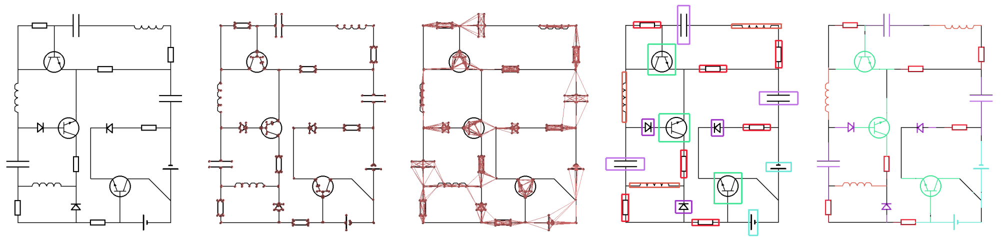


Figure 2.6: ***Application to image data I:*** *Sample circuit diagram (from left to right: input, features, feature graph, detected subgraphs, final symmetries after growing; corresponding instance obtain the same color).*

Triangle meshes: For reverse engineering "perfect" triangle meshes, we place one feature point at the barycenter of each triangle and use the vector of sorted side length as descriptor. As an additional preprocessing step, we detect polygons formed by triangles and retriangulate these consistently, as this tends to be the main source of inconsistency even in "perfect" data. We use the triangle mesh connectivity to create an initial feature graph. Again, we then execute the same graph matching algorithm. Region growing is not necessary in the triangular case.

2.8 Results

We have implemented the proposed algorithm and applied it to a number of benchmark data sets. We have looked at three different cases: Synthetic data, real-world scanner data and data from other modalities.

Synthetic data: We have constructed a data set consisting of a plane with about 50 sketched faces embossed in various orientations (height fields constructed using image editing software). Figure 2.7 shows the result: Every instance of the face is recognized except for the face in the middle that is incomplete in the sense that it shares a double feature with another face, which is not covered in our disjoint symmetries model. Please note that the borders of the detected region resulting from region growing are entirely defined by the contact with other instances and the border.

Next, we have added Gaussian noise with standard deviation $\sigma = \pm 10\%$ of the height field amplitude, which is quite substantial. In this example we used ICP during pattern expansion, as described in Section 2.4.5.

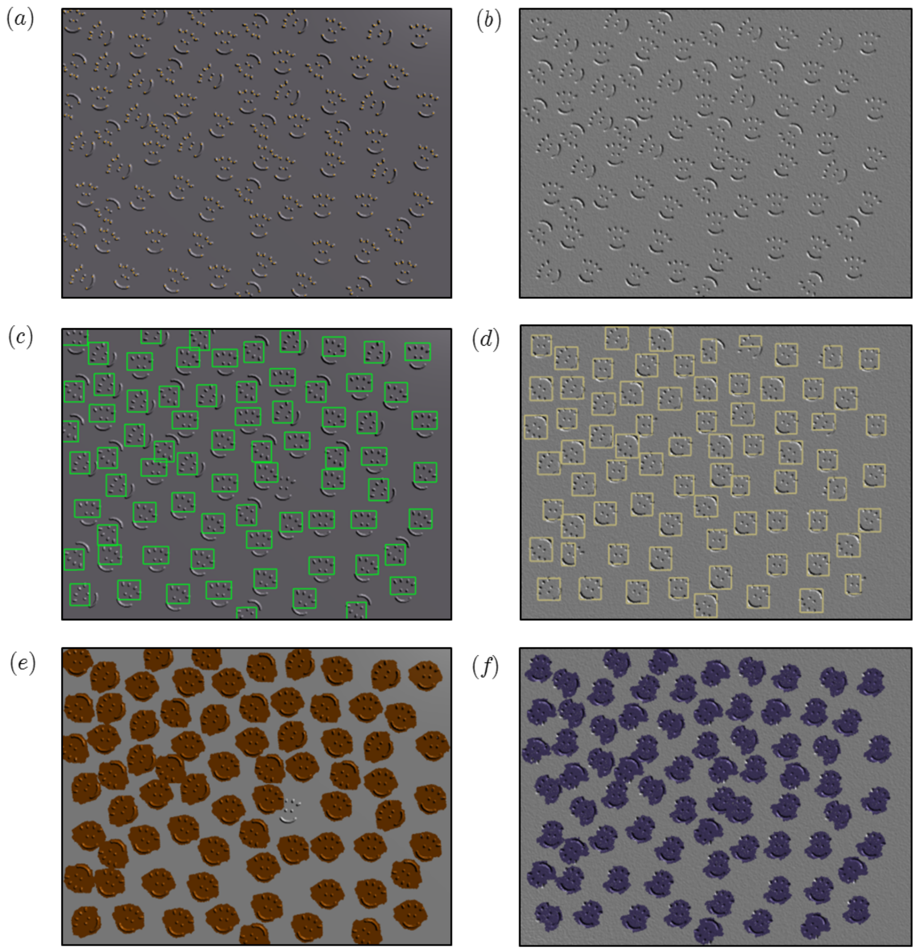


Figure 2.7: ***Faces synthetic data set:*** *top row: features, middle: detected subgraphs, bottom row: symmetry detection result. The left image in each column shows the plain height field and the right image the version with $\sigma = \pm 10\%$ Gaussian noise.*

Actual 3D scanner data: We have tested the approach on three different real-world 3D scanner data sets. The first example shows a historical artifact with multiple engraved figures of equestrians (Figure 2.8). Although the object is man-made and thus shows some shape variation, our algorithm is able to recognize two instances almost completely, up to a small portion at the head of the horse, where the shape variation is too drastic. The third figure on the same piece is geometrically too different under our employed rigid matching criterion so that no correspondence is detected.

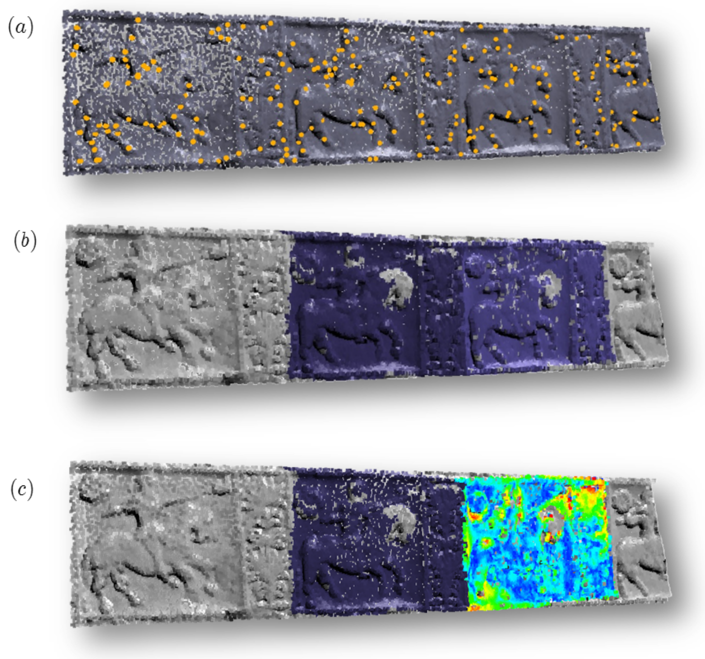


Figure 2.8: ***Engraved horseman (historical artifact):*** *Top: Slippage features, middle: symmetry detection result, bottom: matching residuals (blue = low, red = high). Data set courtesy of Allan Chalmers.*

The second example shows a scan of the "Zwinger" in Dresden, a historical building from the 18th century (Figure 2.9). Similar to the engraved horse, two out of three instances are detected. The third instance is missed due to noise and acquisition holes. Balconies and windows are detected separately.

The third example is a scan of a small clay house model, which has been hand modeled, therefore showing only imperfect symmetries (Figure 2.10). In

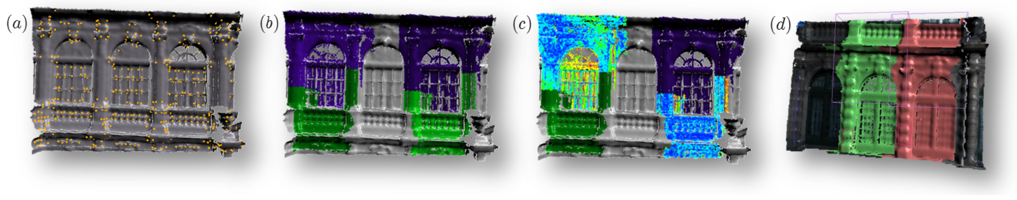


Figure 2.9: ***Zwinger at Dresden*** *(3D scan of the actual building): (a) slippage features, (b)symmetry detection result, (c) matching residual. (d) result on another part of the same building. Data set courtesy of Markus Wacker.*

this example, our algorithm detects all salient symmetries except from three small windows, where the feature detector was not able to provide sufficient coverage. For detecting features on a smaller scale, such as roof tiles, the resolution of the 3D scan is not sufficient.

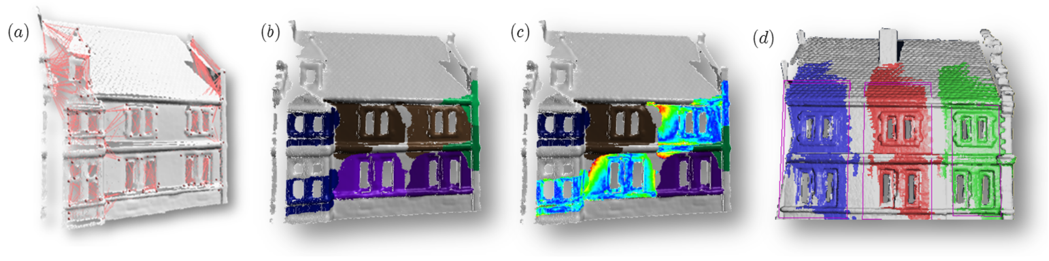
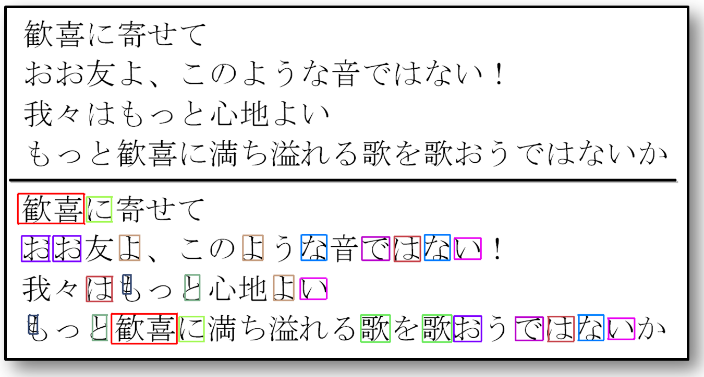


Figure 2.10: ***Clay house model:*** *We tested our method on raw single scans. (a) detected features and graph, (b) symmetric parts (instances are color coded), (c) matching residuals(in comparison to color coded instances), (d) result on another scan of this building (different color coding). The shape variation of the handcrafted clay house windows causes symmetric growing errors on the lower left of the instances.*

Triangle meshes and bitmap images: We have tested the variant of our algorithm for reverse engineering perfect triangle meshes on parts of the well known "power plant" model, which contains a large amount of redundancy but does not provide the original building plan with the instantiation structure. For this clean situation, we were able to get again practically perfect detection results, as shown in Figure 2.12. Application of our algorithm to bitmap images that show recurring sub images also leads to a very good recognition rate. For example, we were able to fully automatically identify the symbols of a circuit diagram (Figure 2.6) and identify recurring phrases in a Japanese translation of the traditional German poem ode to joy (Figure 2.11).



歓喜に寄せて
おお友よ、このような音ではない！
我々はもっと心地よい
もっと歓喜に満ち溢れる歌を歌おうではないか

歓喜に寄せて
おお友よ、このような音ではない！
我々はもっと心地よい
もっと歓喜に満ち溢れる歌を歌おうではないか

Figure 2.11: ***Application to image data II:*** *Japanese text; top: input image, bottom: recognized symmetries*

Computation time: Our algorithm consists of different steps with varying computational costs: The initial feature detection is rather expensive and takes about 10 minutes for the examples shown here. The computation times of the graph matching and region growing steps were in the range of a few minutes in each example,depending on the quality of the input. In the face example without noise, with well distinguishable descriptors, it takes only a few seconds per step. In the noisy example much more time is needed because many ICP tests during pattern expansion are necessary. In our examples region growing on the point clouds takes also a few minutes, plus a couple of minutes to precompute a k-nearest neighbor graph of the original sample points.

2.9 Applications

In this section we demonstrate applications based on the detected rigid symmetry information. The correspondence mappings used in this section were computed by an advanced version of the described algorithm (Bokeloh et al. 2009). In this work, the slippage based feature point detector we applied was extended to identify slippable regions with one degree of freedom, the ”slippage line features”. They can be detected more easily and frequently than the slippage feature points. The slippage line feature based detection method is

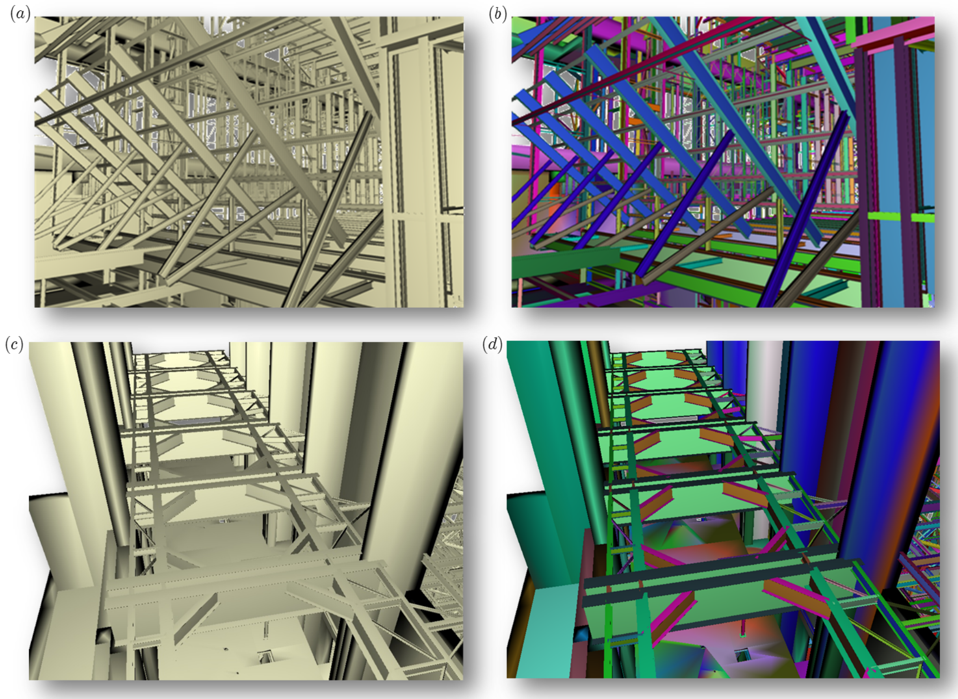


Figure 2.12: ***Application to triangle meshes:*** *All instances of the same type are coded in the same color. For such clean data, we obtain a more or less perfect recognition performance. Data set courtesy of University of North Carolina at Chapel Hill.*

not part of this dissertation. We use only the detected mappings of this work to better demonstrate the benefits of symmetry-awareness in our developed applications.

2.9.1 Simultaneous Editing

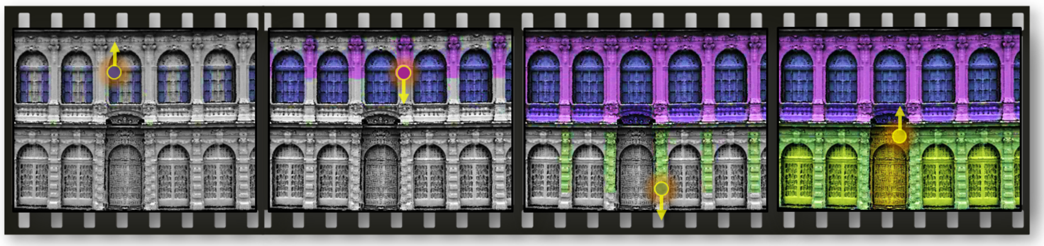


Figure 2.13: *Symmetric drawing on a large building (image cut-out). We employ the found correspondence mappings of (Bokeloh et al. 2009) in a painting application. The user draws on one arbitrary instance (yellow circle) and all operations are mapped to the symmetric instances. The geometry of the facade has already been improved by our symmetry-based reconstruction, see Section 2.9.2.*

Using the known symmetries, every single manipulation by a user, clicking on the model, can be transferred to the symmetric regions. This can be used for every type of manipulation like deformations or modifications of point attributes such as colors, material properties or texture. If the user manipulates an urshape $\mathcal{U}$ of a symmetric part type, we use the computed mappings f_i to transfer the manipulations to all other instances $\mathcal{P}_i$. For user convenience, we allow also to manipulate any instance $\mathcal{P}_i$ and then use f_i^{-1} to transfer the edits to our urshape and then the other instances (Figure 1.3).

In Figure 2.13, we demonstrate this method, directly coloring a huge scanned building with many windows within seconds by a few mouse clicks. This is a very powerful tool to help for example architects working with building scans.

2.9.2 Symmetry-based Reconstruction

A common problem of terrestrial range scans is that they suffer from strongly irregular sampling, noise and acquisition holes due to occlusion.

The sampling rate is not high enough for fine details in some parts of the model, especially in the upper levels of a building, because the scanner is often positioned on the ground. Also parts of the object, that are far away during a scan have a worse signal to noise ratio.

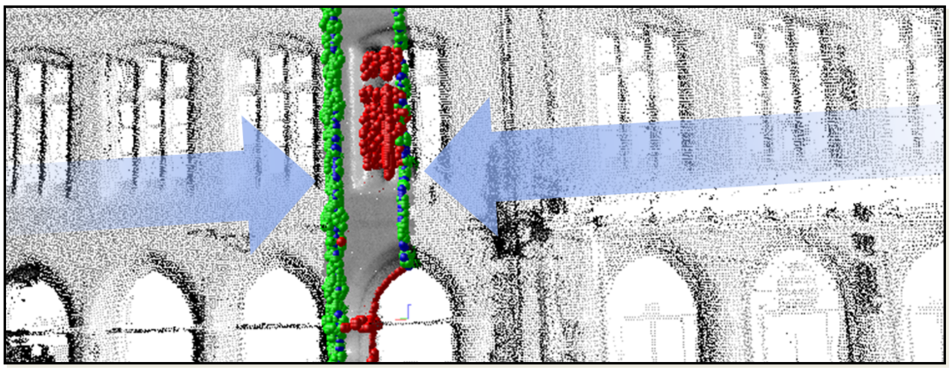


Figure 2.14: *Points of all other instances are back transformed using T_i^{-1}. The Colored dots indicate growing boundaries: Red - geometric error, blue/green - touched other instance.*

Many details can be occluded: In a single scan there are many self occlusions, for example there is only one inner side of a window frame visible for the scanner. In city scanning many occluding structures like trees, traffic signs or parking cars disturb the data. Most of these problems can only be avoided by acquiring and registering several scans with great time efforts.

Our method relaxes this problem. We assume that recurring parts of the scan are of approximately identical geometry, which is true for most examples of modern architecture. So we assume, that we find scanned data points of the same part, e.g. a window several times in one scan acquired from different relative view points. Also each part is sampled at other points of its surface.

Considering this observation, to repair acquisition problems automatically, we propose a reconstruction-by-symmetry algorithm, in the spirit of (Thrun and Wegbreit 2005; Pauly et al. 2005; Gal et al. 2007; Pauly et al. 2008).

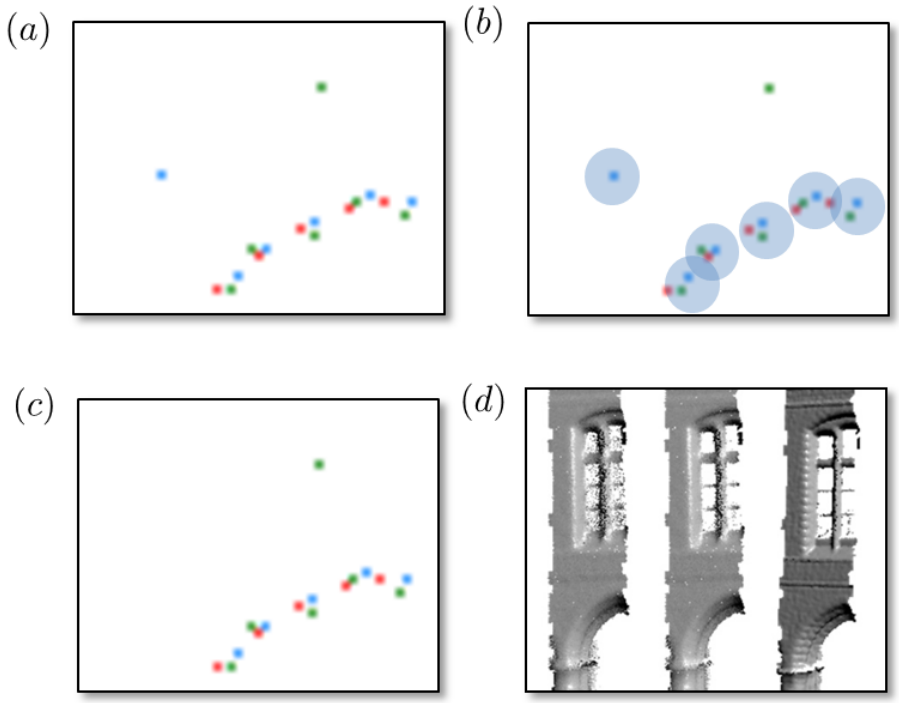


Figure 2.15: *Identifying Outliers: (a) Datapoints of different instances (Instances red, green and blue in this example) form a surface but there are outliers. (b) For the blue instance, we count the number of datapoints of other instances to confirm the surface. (c) Blue instance points with no or little support are removed. (d) From left to right: Raw merged data points, outliers removed, MLS-Projected surface.*

We use the automatically detected similar parts and compute a high density average to improve the sampling quality in regions where such redundant information is available.

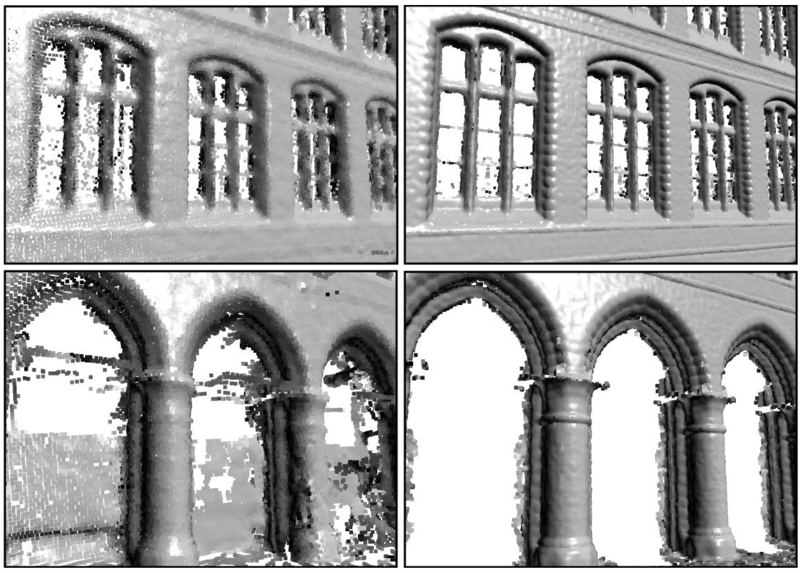


Figure 2.16: *Reconstruction examples for the old town hall data set (courtesy of the Institute for Cartography and Geoinformatics Hannover). Left: Noisy input data, right: result of our symmetry based reconstruction. Using the data of many repeating instances, we are able to reveal a pattern at the window frames that was not visible in the scan before. Symmetry transformations computed using (Bokeloh et al. 2009).*

We start the urshape improvement by mapping all points of all instances $\mathcal{P}_i$ into the urshape, using f_i^{-1} (Figure 2.14). We are taking all points into account that fall into one of the urshape voxel cell under the instancing transformation. From this set, we remove outlier points. Outliers are points that show up in less than 30% of the instances. For this co-occurrence test, we use a small sphere radius around the sample point, according to the scanner sample spacing, and check whether the instance provides such a supporting point within that radius (Figure 2.15). The remaining non-outlier points are then projected on the common surface using a quadratic MLS approximation (Figure 2.15d).

Gathering points of all instances, the sample spacing of this urshape is very high and irregular. Therefore, as a last step, we resample to a lower resolution. This leads to a regular sample spacing.

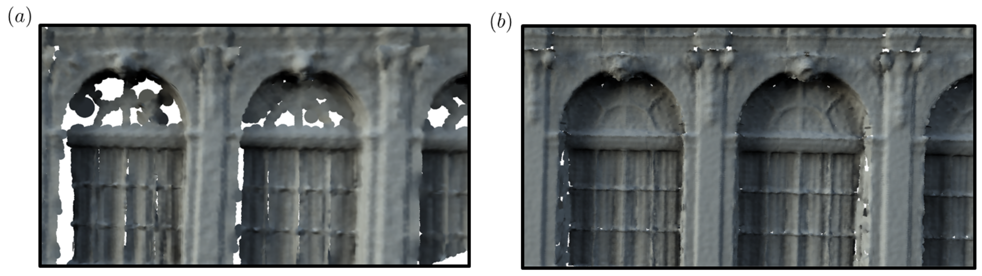


Figure 2.17: *Reconstruction example for the single scan Zwinger data set (courtesy of Markus Wacker). (a) Input scan, rendered with surface splatting. Data is missing inside most of the window frames and upper parts of the window glass. (b) Our symmetry based reconstruction gathers the information of all instances and returns a complete geometry. Symmetry transformations computed using (Bokeloh et al. 2009).*

With this symmetry-based reconstruction technique we achieve super resolution reconstruction results, sharper edges, more details and a regular sample spacing (Figures 2.16 and 2.17).

2.9.3 Compression

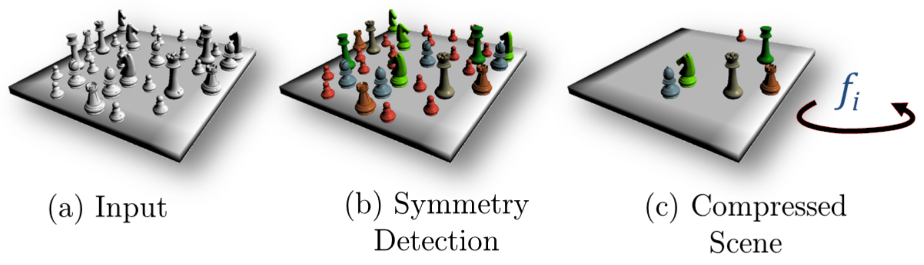


Figure 2.18: *Automatic compression example for triangle mesh input. Similar parts are detected by our method (b). It is now sufficient to store only one instance per part and the transformations.*

After our symmetry-based reconstruction, we copy the improved urshape using our mappings to all instances and discard the former instances. As shown, this step can drastically improve the quality of the scan (Section 2.9.2). Storing this information only for scanned point clouds reduces the amount of data by magnitudes. An example is shown in Figure 2.19: The point count goes down from $2,758,206$ data points on the left side to $22,082$ on the right, which is less than ten percent of the original data size. We save over 90% of data points while improving the visual quality. This can also be employed for rendering and ray tracing point clouds very efficiently.

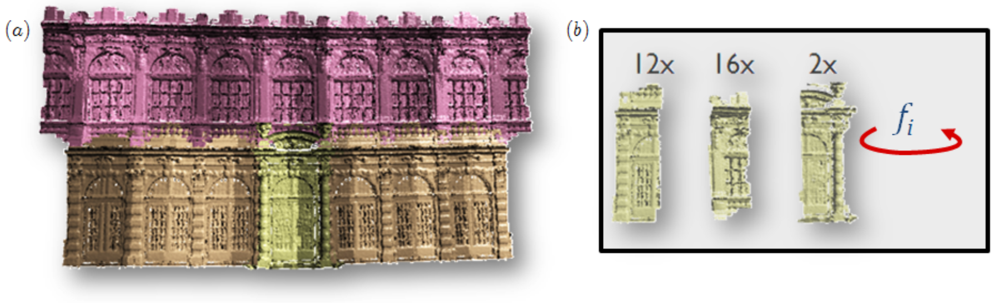


Figure 2.19: *Compression of the symmetry-based reconstructed scanned Zwinger dataset.*

2.10 Summary

In this chapter, we have presented a novel approach to rigid symmetry detection in geometric data. Unlike previous approaches, the new technique is based on a subgraph matching algorithm. The different method has advantages over previous techniques: It does not require the detection of the global symmetry structure prior to handling small scale symmetries and can be generalized to more general matching criteria that cannot be described by voting in transformation spaces.

We deliver a practical proof of concept that such an approach can actually perform reliable symmetry detection, by combining the novel randomized graph matching algorithm with a stable and general feature detection technique and a region growing geometric validation step. Using the known symmetry correspondences, we can simultaneously edit geometry, employ symmetry-based reconstruction and compression.

In the next chapter, we will examine the generalization of this approach to isometric matching, using measurements of geodesic lengths and angles on surfaces instead of rigid Euclidean transformations as graph distance function.

3

Graph-Based Intrinsic Symmetries

In this chapter, in contrast to the rigid transformation mappings shown up to now, we extend our method to match also deformed symmetric parts: First we develop an algorithm for the case of approximately isometric deformations, based on matching graphs of surface feature lines that are annotated with intrinsic geometric properties instead of using feature points. The sensitivity to non-isometry is controlled by tolerance parameters for each such annotation. Using large tolerance values for some of these annotations and a robust matching of the graph connectivity yields a more general symmetry detection algorithm that can detect similarities in structures that have undergone intensive deformations. This approach allows detecting partial intrinsic as well as more general, non-isometric symmetries. We evaluate the recognition performance of our technique for a number of synthetic and real-world scanner data sets.

3.1 Introduction

We address the problem of a *generalized intrinsic symmetry detection*. This means, we aim to find parts in an object that might actually differ by a significant deformation but, to a human observer, would still appear to be of the same kind. We look at two variants of this problem, which are of increasing difficulty: First, we look at strict isometric symmetries, where we can still assume that the deformations of the instances approximately preserves intrinsic distances (Figure 3.1b). Afterwards, we relax our technique to handle cases of more general, non-isometric deformations (Figure 3.1d).

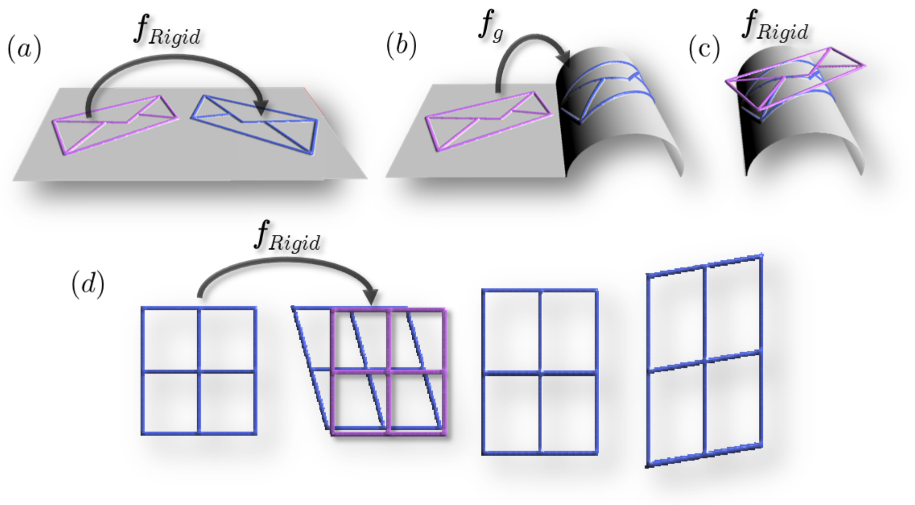


Figure 3.1: *(a) Rigid symmetry detection (b) an intrinsic detection criterion is required, (c) a rigid transformation is unable to overlay the shapes. (d) relaxed graph matching examples with slight variations in length and angles.*

As in the last chapter, our approach is based on feature matching. Effectively, feature extraction transforms the complex, under-constrained geometric matching problem into a simpler discrete graph matching problem. The features provide an abstraction that is much more invariant under typical differences of symmetric instances than the original geometry. Again, only in a second step, to obtain dense correspondences, we perform continuous geometry matching using the matched features as constrained, using a weak smoothness regularizer to interpolate in between. However, the symmetries are actually defined by matching recurring structures of salient features.

In contrast to Chapter 2, where we used slippage feature points, we now employ points and *lines* of *maximum curvature*, corresponding to creases on the surface as feature representation. As the simple feature points only provide not enough information for the higher degree of freedom in the deformable symmetry problem.

This choice of lines and curves is motivated by human perception: for humans, crease lines are particularly salient cues for shape recognition (Kent et al. 1996). This fact is not surprising, as for many real-world objects, a few ridges and valleys already encode most of the geometric information (Ohtake et al. 2004). We annotate the network of feature curves with intrinsic properties of these curves, such as length, intrinsic angles, and intrinsic curvature, which are checked during graph matching. The resulting graph is invariant to isometric deformations of the surface by construction. By relaxing the matching precision for these geometric attributes, i.e. permitting larger variations in lengths, curvature or angles, we obtain a generalized symmetry detection algorithm, that can still detect similar structures in objects that differ by significant deformations.

In order to evaluate the performance in practice, we apply the symmetry detection technique to a number of data sets with symmetric structures of varying similarity, ranging from rigid to not even isometrically similar. Even in the general case, we are able to identify many of the important symmetries that are apparent to a human observer fully automatically.

3.2 Related Work

As explained in the last chapter, the most used group of symmetry detection techniques is based on transformation voting (Mitra et al. 2006; Podolak et al. 2006; Loy and Eklundh 2006; Pauly et al. 2008). The key idea of these techniques is to parametrize the transformation functions with a small number of parameters (such as translation, rotation, scaling), find a set of candidate correspondences and vote for matching correspondences in a Hough space. Transformation voting techniques are currently probably the most frequently used, state of the art class of techniques. However, it is not obvious how to generalize the concept to more complex transformations that require many more parameters which might not be suitable for voting. For example, the parameters of a general free form deformation affect the deformed instances only locally so that voting in a global parameter space is not possible. A similar parametrization problem also applies to geometric hashing (Lamdan and Wolfson 1988; Gal and Cohen-Or 2006). For other approaches, such as robust auto-alignment (Simari et al. 2006), spherical harmonics analysis (Martinet et al. 2006), or primitive fitting (Schnabel et al. 2008), it is so far also unclear how to handle more general transformations.

Only few authors have so far addressed the problem of detecting symmetries under non-linear mappings: (Ovsjanikov et al. 2008) present a technique for detecting global intrinsic symmetries of objects by analyzing eigenfunctions of the Laplace-Beltrami operator of the surface. This approach reveals in a very elegant way global symmetries based solely on intrinsic computations, and is therefore invariant to isometric deformations. However, due to the global nature of the eigenfunctions, it is unclear whether this approach can be used for partial symmetry detection, which is the aim of our approach. (Raviv et al. 2007) detect symmetries by finding feature point sets that preserve geodesic distances followed by a numerical optimization of an isometric embedding by generalized multi-dimensional scaling (GMDS) (Bronstein et al. 2006). This approach does not take into account multiple simultaneous symmetries but rather aims at finding global symmetries in order to detect asymmetric irregularities. Both papers do not consider generalizations beyond isometric matching.

In shape retrieval, topological matching techniques have been used to recognize semantically similar shapes (see for example (Hilaga et al. 2001)). (Zhang et al. 2008) proposed a shape matching technique based on comparing graphs of extremity features and evaluating the induced deformation of a match, aiming at matching shapes such as humans or animals. The technique is able to compute matches between rather different objects, and is a technique that addresses this problem from a global optimization perspective. (Allen et al. 2003) match different human body shapes using local optimization guided by (manually placed) markers, in a formulation very similar to our final continuous matching step. (Tevs et al. 2009) consider pairwise isometric matching of deformable surfaces using graph matching, focusing on robustness in the presence of holes and topological noise in the input surfaces. None of these techniques consider the case of detecting partial symmetries within objects.

3.3 Overview

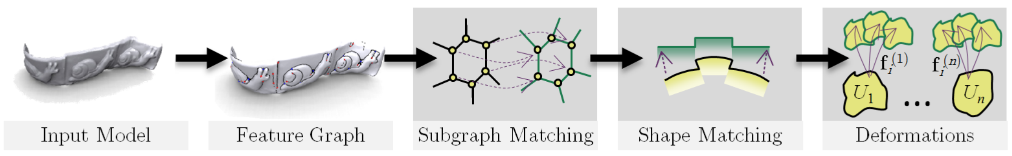


Figure 3.2: *Pipeline - we first detect a network of feature lines on the object surface. Next, we match the resulting graphs, which are annotated with intrinsic properties. By adjusting the admissible tolerance, we can continuously move from isometric to more general matching. The resulting discrete structures are used to initialize a numerical deformable shape matching step, which yields a set of deformation functions between symmetric instances as result.*

We give a brief overview of our technique: The processing pipeline is shown schematically in Figure 4.2. Our algorithm expects a point sampled representation of the manifold as input. This representation allows us to handle both 3D scanner data and triangle meshes, which can be easily sampled in a preprocessing step (we employ a uniform Poisson disc sampling in these cases). The first step of our processing pipeline is the detection of a network of feature lines on the object surface, as detailed in Section 3.4. This yields a graph $G = (V, E)$ of surface lines, each of which is annotated by various geometric properties. In this graph, we then look for recurring subgraphs (Section 3.5). Afterwards, we employ local deformable shape matching (Section 3.6) to assign surface area to corresponding instances and compute dense correspondences, using the discrete matches to guide the alignment. We repeat this algorithm several times until no more significant symmetries are found. As final output, we obtain a *symmetry set* $\mathcal{R} = \{\mathcal{P}_0, ..., \mathcal{P}_n\}$ consisting of n *instance sets* $\mathcal{P}_i$. Each instance is encoded by a single representative "urshape" $\mathcal{U}_i$ and n_i

deformation functions $f_i^{(j)} : \mathcal{U}_i \to \mathbb{R}^3, j = 0, ..., n_j$ that deform the urshape to match the original geometry. These deformation functions establish dense correspondences between all surface pieces within an instance set.

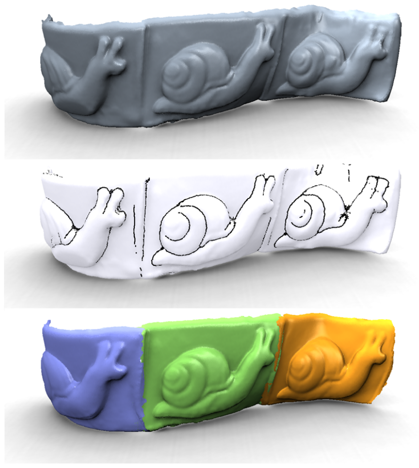


Figure 3.3: *Intinsic symmetries: Snails example. Input geometry, feature lines, found parts.*

3.4 Feature Detection

The goal of our method is to detect symmetric structures in 3D shapes that appear to be of the same type to a human observer, although they cannot necessarily be mapped to each other by a simple transformation such as an affine map. In order to achieve this invariance, we first transform the input data into a discrete feature representation that captures the structure of an object in the connectivity arrangement of these discrete elements.

In Chapter 2, we employed slippage feature keypoints. The slippage based detector was later extended to slippable regions with one degree of freedom, the slippage line features in (Bokeloh et al. 2009). Instead of using these lines for deformable symmetry detection, we switch to another type of line features, namely curvature based feature lines. Our choice is motivated by a purely intrinsic feature approach, originated on Gaussian curvature features (see 3.4.2) and improved curvature feature line behavior in the deformed symmetry problem setting as described later.

For the general symmetry detection problem, we extended the initial intrinsic detector by implementing an extrinsic curvature based feature detector (Subsection 3.4.1) looking for crease lines of maximal principal curvature. The theoretical downside of this approach is that principal curvatures are not isometrically invariant. In practice, this is usually no problem. Real-world variations in shapes are typically smooth such that small scale extrinsic properties are approximately preserved as well. Also, as detailed in (Ohtake et al. 2004), crease lines contain the most important part of the overall geometric shape information and correspond to perceptually important shape cues. This is particularly essential in the case of generalized symmetries: Here, the pattern of crease lines actually *defines* the notion of what is considered to be symmetric, in an attempt to coarsely mimic human perception.

In Figure 3.4 we show a comparison of our extrinsic curvature based feature lines and the slippage-based feature lines of (Bokeloh et al. 2009). Our features were derived as described in the next Section 3.4.1. One problem with the slippage-based feature lines is that lines end near regions where crossings should appear because one-slippability is violated here. Our curvature-based method results in less lines, but clear crossings that are essential for our graph matching algorithm.

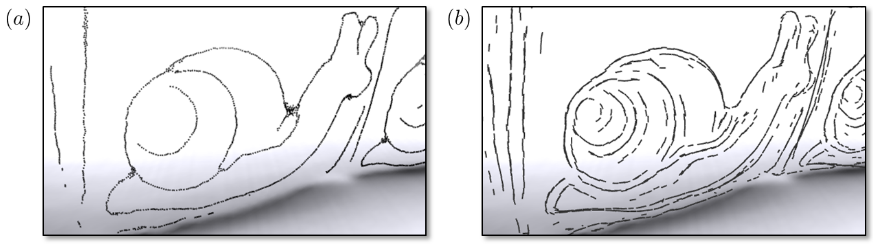


Figure 3.4: *Comparison of line feature types: (a) Our curvature based feature lines, (b) "Slippage line features" of (Bokeloh et al. 2009). Both methods obtain similar results with many more lines for the slippage line features. But less lines are detected near the regions where crossings should appear, because one-slippability is violated there. Our method results in clear dominant lines and crossings that are required for graph matching.*

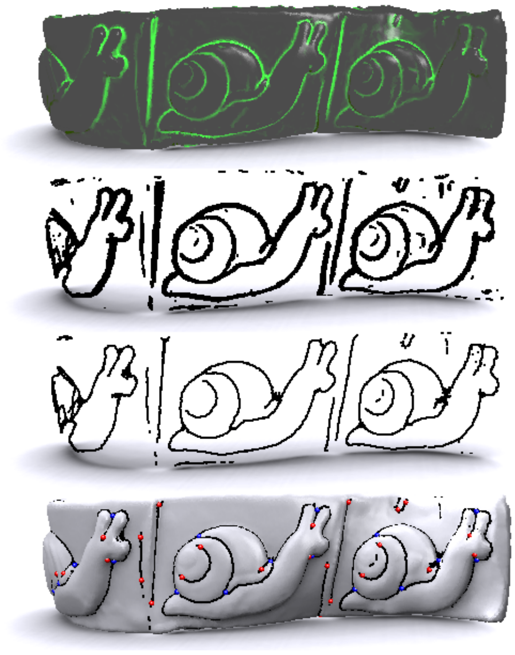


Figure 3.5: *Feature detection pipeline. From top to bottom: product of κ_1 and its second derivative, patches after thresholding, shrunken to lines, resulting feature graph.*

3.4.1 Extrinsic Feature Lines

Our extrinsic feature detection framework was motivated by the "ridges and valley" extraction approach by (Ohtake et al. 2004). In principle, any feature detection algorithm (Gumhold et al. 2001; Pauly et al. 2003; Hildebrandt et al. 2005) could be used in conjunction with the rest of our symmetry detection pipeline. As feature detection is not the main subject of our work, we opt for this simple and easy to implement technique. To get an intuition of typical results, Figure 3.5 shows an overview of the results obtained for the "snails" data set from Figure 3.3.

Feature Scale and Preprocessing: In the following, we describe how to extract features at a fixed scale ϵ_{scale}. Later, we will vary this scale in order to build a multi-resolution representation. We assume we are given a sampled representation of a surface $\mathcal{S}$. First, we resample the point set by deleting points that are closer to their nearest neighbors than $0.25\epsilon_{scale}$. This step makes sure that the costs for the subsequent computations do not become too large. Afterwards, we form a connectivity graph that connects each point to its 12 nearest neighbors (excluding edges longer than $3\epsilon_{scale}$, to make the construction robust to outliers). This graph encodes the apparent connectivity of the manifold and is used in all subsequent operations. Later in the algorithm, a subset of this graph will form the curve lines.

Curvature computation: Given a point $\mathbf{p}$, we consider the curvature tensor $\mathbf{C}(\mathbf{p})$ and its two eigenvalues κ_1 and κ_2, sorted by their absolute value. We estimate these quantities via a quadratic moving least squares (MLS) fit (Alexa et al. 2003; Douros and Buxton 2002), using a Gaussian window of size ϵ_{scale}. In order to make the resulting values scale invariant, we normalize the local surface pieces by scaling by ϵ_{scale}^{-1} before MLS fitting. A crease line should maximize the larger of the two principal curvatures κ_1. In particular, this implies that the first derivative of κ_1 vanishes in the direction of $\mathbf{t}_1$ while the second derivative in this direction is non-zero. We compute this second derivatives by again fitting a second order MLS approximation to the values of κ_1 computed previously. We prefer this technique as it is more robust than attempting to directly fit a 4th order representation (the number of parameters to be estimated is much smaller, reducing overfitting).

Classification: We now classify candidate crease line points by looking at the product of the absolute value of the first principal curvature and its second derivative in the principal direction. We threshold this quantity to obtain a set of curve line candidates. In practice, the product of these two quantities is a quite robust measure for crease line detection: Even varying the threshold value by a factor of 10 or 100 does not change the solution drastically. In contrast, a direct thresholding of the curvature values is very sensitive to parameter setting; even changes by a factor of 1.1 can lead to substantially different results.

Feature line shrinking: As a result of the classification, we obtain narrow patches of crease line points, connected by the connectivity graph. In order to obtain smooth, linear curves, we apply a very simple filtering technique: For each point, we collect all neighboring points within a graph distance of no more than $12\epsilon_{scale}$ and move each point to the centroid of this set. The centroid is then projected back on the manifold using again a quadratic MLS surface

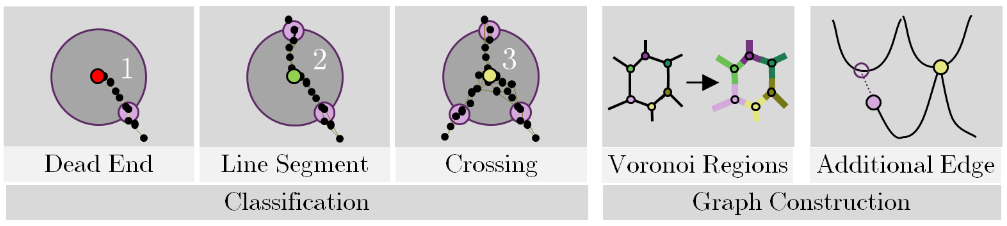


Figure 3.6: *Steps for building the feature graph. First, intersections and dead ends are detected as feature points. Second, Voronoi regions on the feature lines are computed around all feature points in order to form a connectivity graph - when two Voronoi regions meet, their graph vertices are connected by an edge. Lastly, small gaps are closed to make the algorithm more robust.*

fit. This smoothing is rather simplistic and tends to oversmooth highly curved lines; however, as it does so consistently, this is no problem in the context of our application.

Feature line graph: The next task is to convert the set of feature line points into a graph of lines and crossings of such lines. We employ a simple sphere crossing test: For each point on a feature line computed previously, we center a sphere of radius $5\epsilon_{scale}$ around it and determine all pairs of connected curve points for which the connecting edge in the connectivity graph intersects with the sphere. If multiple such intersections occur within a distance of ϵ_{scale}, they are counted only once. This test yields a classification of curve points into line segments (2 intersections), dead ends (1 intersection) and crossings (3 or more intersections, shown in Figure 3.6). For dead ends and crossings, each connected component of the same type is converted into a vertex in the graph. In order to form the graph edges, we simultaneously run Dijkstra's algorithm starting from all vertices in order to compute Voronoi regions in the graph of line feature points. Whenever two Voronoi regions meet, their graph vertices are connected by an edge. The length of this edge is set to the according Dijkstra distance. In addition, we estimate the average integral geodesic curvature of the edge by fitting an MLS quadratic curve to each point along the Dijkstra path and averaging the values. We will later use this measure to qualitatively distinguish edges by their bending direction (left/right w.r.t. the travel direction), which is often at least qualitatively preserved even in general, non-isometric symmetries. We also compute the tangent vectors to the curve lines at the graph vertices, using finite differences. They will later be used to check intrinsic angles.

Additional edges: In practice, the line feature detector is not always able to correctly detect intersections of line features, in particular if the height or depth of the crease lines is rather shallow. Even if the crease line actually vanishes before hitting another line, the human visual system would take the logical continuation into account and use this property for feature matching. Consequently, we detect "dead end" feature points that are close to other feature lines and extend their course in tangential direction, hitting the other line with a newly created intersection feature (see Figure 3.6, right).

Feature line parameters: As explained, the feature detection and graph building steps rely on many parameters. We set these parameters empirically and use the same default settings in all examples. Small inconsistencies in the resulting feature graphs are handled by our graph matching pipeline (Figure 3.7). But changing feature graph parameters significantly results in unreasonable feature graphs without any recurring patterns to detect.

3.4.2 Intrinsic Features

For purely intrinsic symmetry detection, we replace extrinsic pairs of principal curvatures by the intrinsic Gaussian curvature. We use the same MLS scheme for the computation, just multiplying the two principal values. We need to modify this scheme slightly to make it reliable in practice: Quadratic estimations with radially symmetric Gaussian weights can lead to a bias when being applied to a patch of finite size. Straight crease lines of complex cross section, but obviously with zero Gaussian curvature everywhere, may appear to have a phantom curvature in the direction of the crease line. This is because a spherical cut-out area might be best approximated in a least-squares sense by a quadratic polynomial that bends into two directions and sampling limitations prevent us from converging to an unbiased solution. We avoid this problem by using, in a second step, an unweighted square window with axes aligned to the previously computed directions of curvature.

The maxima of Gaussian curvature do not form line structures but point features. Therefore, we form our graph differently: First, we extract point features thresholding the Gaussian curvature and connecting the resulting peaks to their k nearest neighbors with geodesic paths on the surface (we use $k = 8$). We annotate each point feature with the sign of the Gaussian curvature (hyperbolic, spherical). Geodesics are again estimated using Dijkstra's algorithm on the connectivity graph. The output (graph of lines with crossings) is the same as of the first technique and use in the same way in the subsequent pipeline. However, this intrinsic graph provides less information than the extrinsic feature lines. In particular, the intrinsic curvature vanishes everywhere on the geodesic lines. However, we will only use it in the case of strict isometric matching, where we do have the stronger invariant of approximate distance preservation (which is not available in the general case of large non-isometric deformations).

3.5 Intrinsic Feature Graph Matching

The main component of our algorithm is a graph matching algorithm that detects similar feature constellations. The matching routine consists of two nested loops: The inner loop is a randomized greedy algorithm that generates random candidate solutions. Depending on the random choices made, different solutions (i.e., instance sets) of different quality will be output. The outer loop then iterates this routine several times in order to sample the solution space and only outputs the best instance set found. We can then apply this algorithm repeatedly, removing graph elements from already identified instance sets, to output all symmetries within an object.

3.5.1 Inner Loop

We assume we are given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of feature points and lines, corresponding to vertices and edges. Each such discrete element might be annotated by continuous geometric properties. Our task is to find a subset $\mathcal{U}_{\mathcal{G}} \subseteq \mathcal{G}$ of the feature graph and a set of discrete mapping functions $f_{\mathcal{G}(i)}$ that map vertices and edges of this subset to corresponding subgraphs $\mathcal{U}_{\mathcal{G}}^{(i)}$ that have approximately the same structure. This means, that the graph connectivity itself should be similar and the annotations should be approximately preserved. By varying how strictly these geometric quantities are preserved, we can go from isometric to more general symmetry detection.

Start edge: Each greedy matching step starts by choosing a random edge from $\mathcal{E}$, which we will call *start edge* e_{start} in the following. The corresponding instance, which up to now contains only this single edge, will be called *start instance* $\mathcal{I}_{start}$. We now look for more instances that are similar to the start instances by comparing the start edge to all other edges e' in the graph. We denote this comparison function as probability $P(e, e')$ (we will give more details on the implementation of this comparison function later, in paragraph "edge matching"). It is important to take into account that instances might be symmetric to themselves, resulting in overlapping but non-identical matches. Therefore, we distinguish directed edges (i, j) and (j, i). This corresponds to a rotation by 180°. In order to also account for mirroring, we tag each instance with a "mirrored" bit, that indicates that the whole instance has changed its orientation (i.e., the cross product of a tangent frame flips its orientation with respect to the extrinsic surface normal). In the worst case, each start edge can appear three more times as starting matches, varying by rotation and mirroring.

Instance growing: So far, we have only a small graph of two vertices and one edge that corresponds to a number of other such small graphs. Our task is thus to extend this with additional matches. This is done by first choosing a random vertex on the start instance. We then examine all outgoing edges $e_{candidate}$ that are not yet contained in the instance and evaluate how well they can be matched to the other instances. We then take the best such match, and reevaluate the other edges until no more matching edge is found. In order to evaluate an edge $e_{candidate}$, we go through all other instances and compute the best matching edge, i.e. an edge e_{match} that maximizes the matching probability $P(e_{candidate}, e_{match})$. Very low scores (below 0.2) are ignored. For each edge $e_{candidate}$, we compute the expected benefit, which is the sum of correctness probabilities $P(e_{candidate}, e_{match})$. Finally, we select the edge $e_{candidate}$ with the highest benefit and all its associated matches.

Selecting the solution: The iteration outlined above stops automatically once no more edges with sufficient matching score are found. After the iteration, we have a situation in which we know a number of correspondences between the start instance $\mathcal{U}_{\mathcal{G}}$ and several other subgraphs $\mathcal{U}_{\mathcal{G}}^{(i)}$. Each correspondence is typically partial, covering a subset of $\mathcal{U}_{\mathcal{G}}$ each. For now, this is no problem; the continuous shape matching step will later automatically determine a consistent common subset of geometry for each instance set. However, we need to remove spurious matches such as cases, in which only the start edge matched and nothing else. We do this by deleting all $\mathcal{U}_{\mathcal{G}}^{(i)}$ that are not supported by at least a minimum number k_{min} of vertex correspondences. The rational behind this strategy is that we need to gather some "evidence" that the match is reasonable. A very small number of correspondences might be correct plainly incidentally, while a larger number makes the match more believable. k_{min} is a user parameter that trades-off false positives versus missing some matches. If we have sufficiently strong geometric evidence, it is often sufficient to use rather small values such as $k \geq 4$.

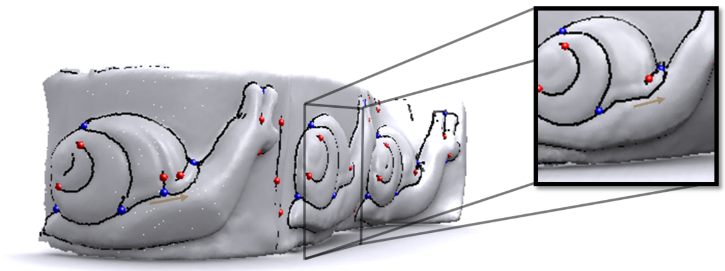


Figure 3.7: *Robustness to graph connectivity differences: If a matching line fits in every criterion but the length, we check to skip an intersection point and compare the extended edge instead.*

Robustness to graph connectivity differences: A feature graph for real-world data sets is usually far from perfect. Therefore, we have to deal with connectivity noise in the graph, which will lead to false positive intersection nodes. This effect can occur for example if noise is converted into small line segments that form crossings that are not present in symmetric instances (Figure 3.7). Hence, we do not check only one outgoing edge but allow the algorithm to skip over intersection points. In case the correct matching edge is inter-

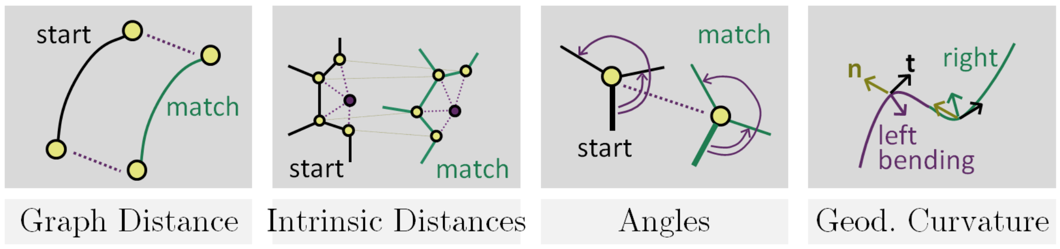


Figure 3.8: *We employ four geometric validation criteria plus a check of the graph structure (subgraph isomorphy). By setting the matching sensitivity for each of these criteria separately, we can generalize the algorithm to non-isometric matching. Typically, we obtain good results in practice by disregarding distances completely and using coarse penalties for angles and curvature.*

rupted by a spurious intersections, the algorithm will try to go up to three points further, along outgoing lines with best matching tangential direction. This feature skipping is performed on both the start instance and matching instance side (conceptually, this is equivalent to just checking additional edges on both sides).

Edge matching: Finally, we need to design our edge matching function $P(e, e')$. We formulate it as a probability value between zero and one. The actual value will not only depend on the pair of edges themselves, but also on the two subgraphs that contain these edges and have already been matched previously. The matching probability is a product of five separate validation scores:

- **Graph structure:** If the two matching feature points do not have the same outdegree, they are unlikely to be a good match (not impossible, because there might be connectivity noise). We also check if a newly matched vertex is connected to a vertex that is already in the instance. In that case, the connecting edge must be present in both instances and connect to the same vertices (thus enforcing subgraph isomorphy). If any of this fails, we multiply the probability by a factor of 0.25.

- **Edge length:** The newly inserted edge should be of the same length. We assume Gaussian noise with a user specified standard deviation σ_{length} and output the corresponding probability density, normalized to the range $[0 \ldots 1]$.

- **Geodesic distances:** For this criterion, we compute the geodesic distances on the *original* surface $\mathcal{S}$ of the newly inserted feature point to all other feature in the start instance and the matched instance (again, using Dijkstra's algorithm). We again assume Gaussian noise with standard deviation σ_{geod}, and multiply the resulting probabilities for the deviation and output a normalized value.
- **Intrinsic angles:** We use an additional criterion where we check the angles of the tangents of the outgoing edges in the tangent plane with respect to the incoming edge. Again, strictness is controlled by a standard deviation parameter σ_{angle}. For the angle matching, we need to check the orientation bit of the instance and reverse the orientation of the angles in cases of reflected instances. The angle criterion is particularly useful in generalized matching scenarios. Angles are often preserved even if distances are not (in object classes such as windows, for example).
- **Geodesic curvature:** Finally, we also check the average geodesic curvature, parameterized by a standard deviation parameter σ_{curv}. Again, a qualitative match is useful for generalized symmetry detection (such as to distinguish between left-bending and right-bending curves). Again, we need to check the mirroring bit to reverse the bending direction in mirrored instances.

3.5.2 Outer Loop

The success of the inner loop depends on a number of random decisions. We therefore execute the procedure repeatedly to obtain several alternative instance sets, sampling the solution space. From these, we chose only the best solution. In order to quantify what the best solution is, we look at the complexity of the result; more complex matches are less likely to be spurious: Given n instances with n_i graph elements each, we assign to this solution a score of: $\prod_{i=1}^{n}(1 - exp(-\lambda n_i))$ where λ is a user parameter in the range of $0.5 \ldots 0.1$ (typically: 0.2). The ratio behind this heuristic is to compute the expected gain of the solution, where each instance is worth one unit and the probability of being a false positive drops exponentially with the number of graph elements used to validate the match (basically naively assuming independent and identically distributed probabilities for each validation step).

3.6 Deformable Shape Matching

Discrete shape matching gives us only sparse correspondences between symmetric instances. In a subsequent step, we propagate this information to the full geometry. As we have already solved the global combinatorial assignment problem, we can now use a locally convergent numerical shape matching technique. For strictly isometric symmetries, an isometric embedding such as the GMDS technique of (Bronstein et al. 2006) would be a good choice (and straightforward to integrate into our framework). As we are also aiming at detecting more general symmetries, we opt for 3D thin-plate splines as introduced by (Allen et al. 2003) for matching shapes with irregular variations. This regularizer does not preserve isometries exactly but, in practice, yields visually good solutions with the ability to handle general shape distortions. In addition, it gives a canonical extrapolation of the deformation field into the space surrounding the deformed object, which is useful for many applications.

We compute a deformation function $\mathbf{f} : \Omega \rightarrow \mathbb{R}^3$ that maps from a volumetric region Ω enclosing the undeformed shape into three space. The thin plate spline energy tries to keep local deformation gradients similar, i.e. minimizes the Hessian matrix of the deformation function (Allen et al. 2003; Brown and Rusinkiewicz 2007). In addition, we add a constraint energy for shape matching, which leads to the following objective function:

$$E(f) = \int_{\Omega} \|\lambda_{reg}\mathbf{H}_{\mathbf{f}}\|_F^2 + \sum_{i=1}^{n_{pos}} (\mathbf{f}(\mathbf{x_i}) - \mathbf{y}_i)^T \mathbf{M}_i (\mathbf{f}(\mathbf{x_i}) - \mathbf{y}_i)$$

$\mathbf{H}_{\mathbf{f}}$ denotes the Hessian matrix of $\mathbf{f}$, λ_{reg} is a user parameter that controls the flexibility of the mapping, $\mathbf{x}_i$ and $\mathbf{x}'_i$ are positions in Ω at which the positions $\mathbf{y}_i$ and Jacobian matrices $\mathbf{Y}_i$ are prescribed, respectively. $\mathbf{M}_i$ are error quadrics that describe the weighting of the position constraints in all spatial directions.

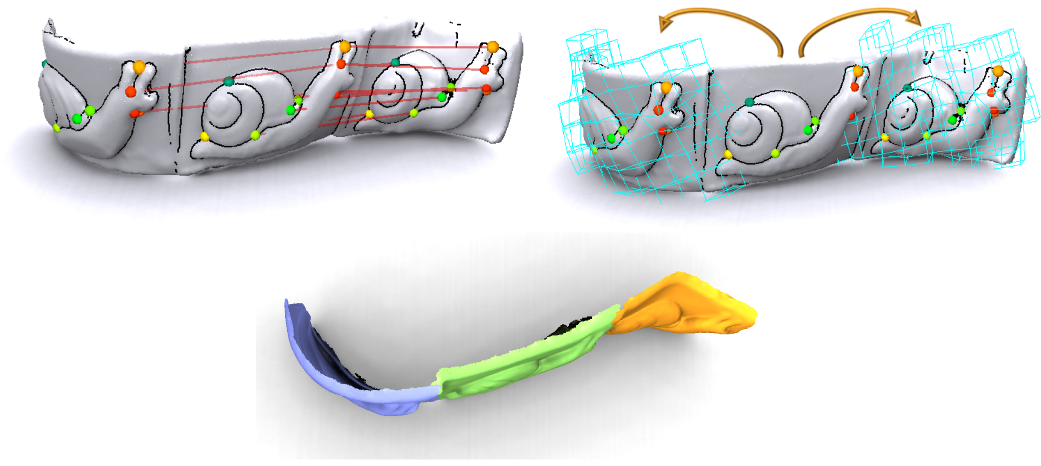


Figure 3.9: *Deformable shape matching the snails example. Top: Visualization of the discrete vertex correspondences of our graph matching step. Middle: Continuous deformation using thin-plate splines. Bottom: A view from the top, to show the non-rigidity of the deformation.*

We solve the variational problem by discretizing **f** on regular grid of basis functions (as we will have numerous constraints, this is more efficient than using globally supported radial basis functions of fundamental solutions). We use a sparse grid that has only entries near actual surface points: The grid is chosen such that the domain is overlapped by all possible basis functions with non-zero support and at least one more additional layer of grid cells to allow for extensions into the nearby volume. The derivatives are approximated as symmetric finite differences of adjacent grid cells. As basis functions, we use simple but efficient piecewise linear functions during the iterations of the deformable alignment and a more costly smooth interpolation with radial Wendland functions in the final iteration where all correspondences have already been established. The discretization leads to a linear system of equations that we solve using a standard conjugate gradients algorithm.

Using this machinery, we can implement a deformable ICP algorithm: We start by constructing the undeformed parametrization domain (urshape) by cutting out the bounding cube of the matched features out of the start instance point set. Next, we setup position constraints that map intersection points to the positions on the matching shape (with identity quadrics $\mathbf{M}_i$). We also map lines to lines using a proportional mapping with respect to arc length as initial guess. With this initial guess, a first alignment is computed. Afterwards, we compute for each deformed point the closest point on the target surface and setup a point-to-plane constraint that attracts the deformed point to the closest target in normal direction. This is implemented by setting the error quadric to a rank one tensor given by the outer product of the normal to the target surface with itself. For each constraint, we also add a copy displaced by the surface normal scaled by the grid size. This helps in keeping the volumetric deformation problem well constrained. We solve the resulting linear system and iterate the deformable ICP until convergence.

We use this to align all instances to the starting instance, which becomes our urshape. After completion, we use a simple region growing algorithm to cut out the symmetric geometry: We start at a random point close to the start edge and in a breadth-first marching algorithm simultaneously extend the covered region. This leads to a growing front of constant geodesic distance that eventually either collides with other fronts or reaches a region with large point-to-plane residuals where the growing stops because of mismatching geometry. To make this outlier robust, we do not stop if less than 20% of the instances are affected by such a collision or a geometry mismatch. If growing reaches the boundary of the initially cut out domain before being stopped, the domain is extended and the iteration continued.

3.7 Implementation and Results

We have implemented our symmetry detection algorithm in C++ to evaluate the performance in practice. The results were obtained on PCs equipped with 2.4 GHz Intel Core 2 Duo processors.

Deformed windows: Our first test data set is a synthetic geometry of four Gothic windows, one of which is folded onto a cylinder, yielding a mapping very close to isometry (Figure 3.10). The discrete graph matching identifies the symmetric windows correctly, including the reflective symmetry. For this rather simple case, we obtain almost perfect correspondences.

This test scene also shows, that our implementation is capable to identify rigid symmetries as introduced in Chapter 2, extending the presented possibilities: The rigid transformation method discussed in Chapter 2 detects three windows only, but not the cylindrical deformed instance.

In a second step, we make this problem more challenging by applying a global non-isometric deformation to the scene (see Figure 3.12). In this case, the generalized symmetry detection algorithm is still able to find the four instances of the window, including reflective symmetries that lead to 8 overall instances. There are small artifacts in the corners of the instances; in this region, the deformable ICP was unable to extrapolate the correspondences precisely.

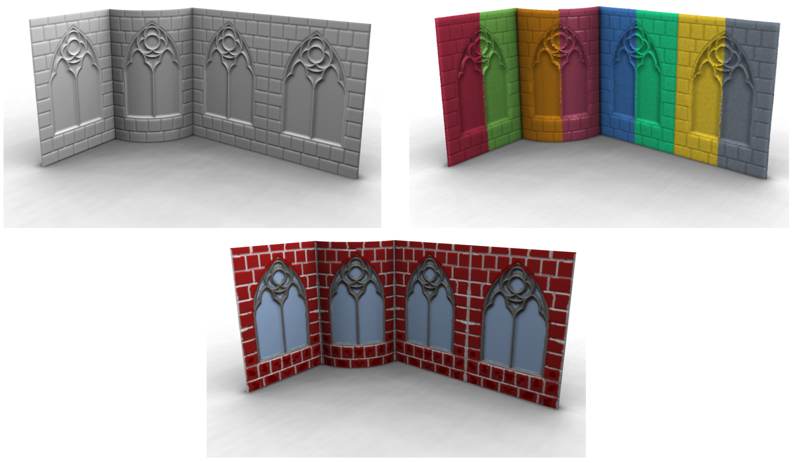


Figure 3.10: *A synthetic example for partial isometric symmetries; left: original geometry; middle: detected instances after region growing; right: painting on one instance and transferring to all others.*

Plasticine snails: In order to have a real-world data set with approximately isometric symmetries, we have modeled and scanned a physical 3D model ourselves. The model consists of a sheet of modeling clay in which three figures of snails have been impressed using a plastic cast. Subsequently, the modeling clay sheet was bent to an S-shaped cross section. The data set has been acquired with a Minolta laser triangulation scanner, aligned using deformable ICP and postprocessed manually, which includes outlier removal, hole filling and MLS filtering for noise suppression. The result is shown in

Figures 3.3 and 3.9. The resulting data set still has several imperfections: for example, the antennae of the head of the snails are flattened out in two instances as well as the back of the leftmost snail (due to falling on the floor before scanning). In addition, the surface shows scratches from sculpting that create outlier lines. These issues have intentionally not been fixed to obtain a realistic test case. After adjusting some parameters (see discussion below), we obtain a good matching results.

Comparison of matching strategies: In Figure 3.11, we compare different matching strategies; the top row shows a relief with three instances each that are rigidly displaced on the left, approximately isometrically deformed in the middle and strongly distorted (using manually placed constraints on a thin-plate spline energy) on the right. We then first run the advanced version of our rigid symmetry detection technique (Bokeloh et al. 2009), which represents the state-of-the-art for rigid feature based symmetry detection. As expected, the technique is not able, even with optimized parameters, to identify the deformed instances. However, the rigid instances are detected reliably. Next, we employ our isometric matching pipeline using intrinsic features based on Gaussian curvature (all other examples use the extrinsic features) and strict threshold on all intrinsic geometric quantities. As a result, the isometrically deformed parts are recognized reliably, but the technique fails for the more general cases. These are recognized by running our approach in "generalized settings" (using extrinsic crease curve detection). Please note that the recognition quality for the simpler cases does not suffer visibly from the generalization here.

More general symmetry examples: We have tested our algorithm on two more example data sets that contain generalized symmetries. The first is a piece of a castle data set, also used (for comparison) in (Bokeloh et al. 2009), see Figure 3.13. The rigid matching technique recognizes just a global reflective symmetry for this data set, because the three gates are of different cross section. The generalized technique actually recognizes that these three instances are structurally similar, including the reflective symmetry within each gate.

A second example is a piece of the back of the Stanford dragon, which features structurally similar scales that are easily recognized by a human observer. However, the actual geometry is varying drastically so that this similarity is

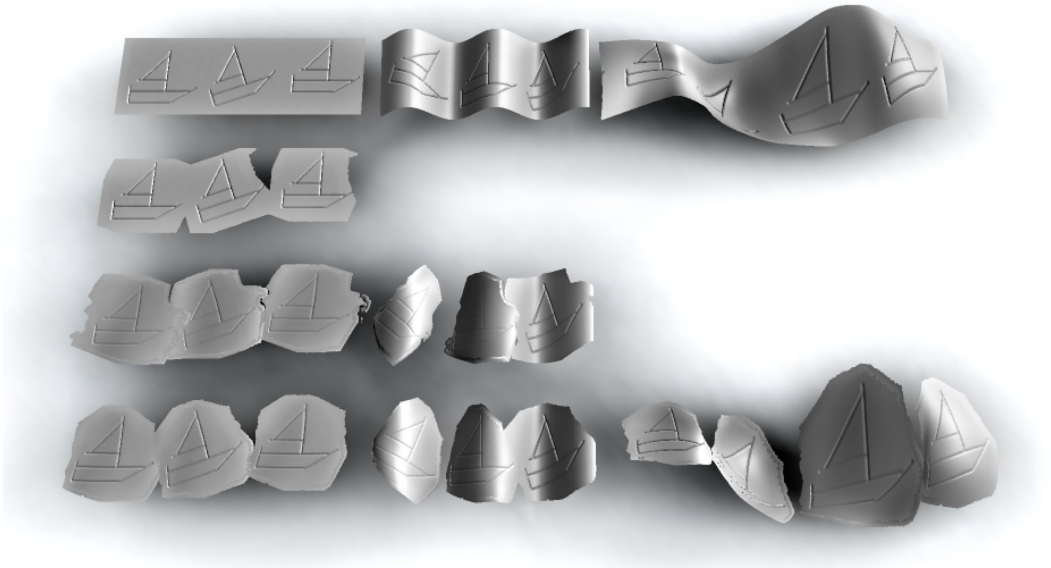


Figure 3.11: *Comparison of different symmetry detection algorithms. The top row shows an example of a relief with rigid symmetries (left), near-isometrically deformed symmetries (middle) and non-isometrically deformed symmetries (right). The second row shows the resulting symmetric instances detected using a rigid symmetry detection technique (Bokeloh et al. 2009). The third row shows our results, for the purely isometric pipeline (Gaussian curvature features, strict preservation of intrinsic distances). The last row shows the results for the generalized symmetry detection scheme (relaxed geometric matching, extrinsic crease feature lines). As expected, the rigid technique fails on deformed examples.*

very hard to detect for an algorithmic feature detector. The connectivity of the boundary line features of the scales varies to strongly from instance to instance it is not possible to find repeating patterns for the boundaries. However, instead of failing our algorithm used the E-shaped notches in the middle of the scales to identify the repeating structure, while the deformable shape matching aligned the boundaries correctly. In the end, our technique is not able to detect all scales but at least detects the majority of them and cuts out reasonable instances automatically by the deformable alignment (see Figure 3.14).

Running time: The most expensive parts of our computation pipeline are the feature detection step (in particular, the MLS computations, which per-

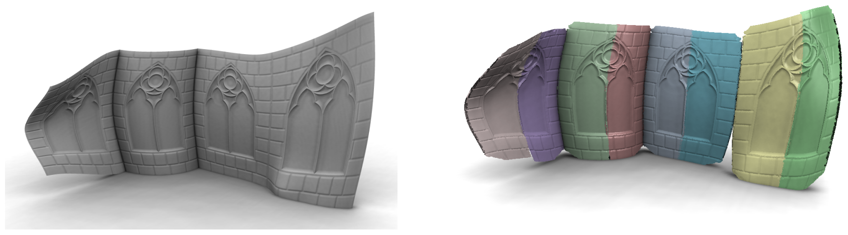


Figure 3.12: *Relaxing partial isometric symmetries. Left: Original strongly deformed geometry. Right: Detected instances after region growing; the image shows the result of deforming the urshape (half a window) by the computed continuous mapping function.*

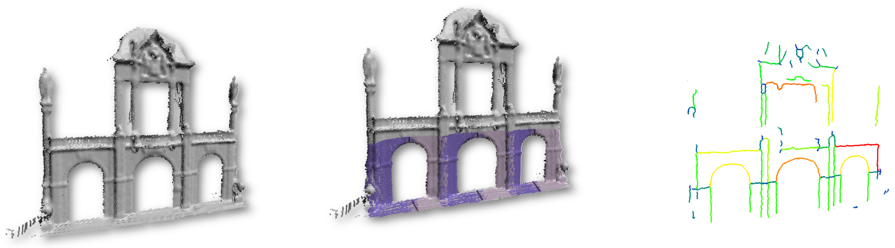


Figure 3.13: *A simple example for using general symmetries on a scanned architectural data set: One of the three gates is of different proportions; our technique recognizes the symmetry, including reflective in each piece symmetry. The rigid technique of (Bokeloh et al. 2009) only recognizes a single global reflective symmetry. The right image shows the detected feature graph (edge lengths are color coded).*

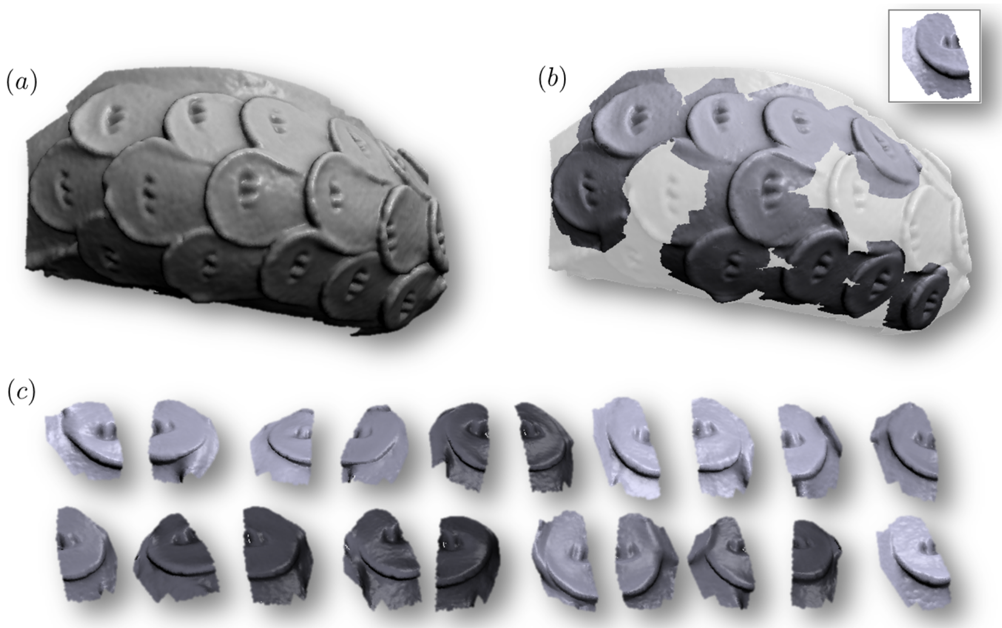


Figure 3.14: *Results for a side piece of the Stanford Asia Dragon data set. (a)Input. (b) Our algorithm correctly identifies more than half of the repeating scales. Due to the strong geometric and even graph connectivity variation, this is a very hard problem. (c) Shows all deformed urshapes, which are very close in shape to the original geometry and match salient feature lines correctly.*

forms many nearest neighbor queries) and deformable alignment. Both require computation times in the range of a quarter of an hour. In comparison, graph matching is inexpensive, with running times of less than a minute for all our data sets. Most of this time is spend on building the feature graph and computing geodesic distances. The core graph matching routine always completes within a few seconds.

3.8 Discussion and Limitations

Our method is able to identify isometric symmetries and establish dense correspondences. For more general symmetries, it is still able to retrieve matches a human observer would expect. However, we do not obtain perfect results, due to the difficulty of the general matching problem.

As most feature-based algorithms, the technique is parameter dependent. Feature detection parameters handling difficult scanned inputs require some time to empirically find working parameters for e.g. thresholds, smoothing radius and curve shrinking parameters. Some of these parameters could be set automatically by analyzing the model or removed by an advanced method for line feature creation.

By design, the important parameters of the graph matching algorithm are the four standard deviations $\sigma_{length}, \sigma_{geod}, \sigma_{angle}, \sigma_{curv}$. These parameters can be used to adapt the matching algorithm to the specific situation. For general matching, filtering by angles and geodesic curvature (with increased standard deviation, though) might still yield good results while preservation of lengths is only an option for near-isometric cases. The only additional parameter we currently need to set is the minimum instance size to filter out small outlier instances; this value is not determined automatically with the current strategy. For region growing and deformable matching, we use a fixed parameter set for all cases.

Besides parameter dependence, our algorithm is mostly limited by the type of features we build our discrete structure on. The choice of high curvature lines works well in many data sets, but currently fails if a surface contains only closed curves without intersections (e.g. Figure 4.18). However, refining the feature detector with additional cues (such as curve corners, slippable regions, geometric primitives etc.) and integrating them in the matching pipeline is straightforward.

3.9 Summary

We have presented a new algorithm to detect general isometric symmetries in 3D shapes. Our method can handle partial isometric symmetries, as well as more general symmetries where the preservation of intrinsic geometric quantities can be relaxed. Using our approach, we are able to detect complex symmetry patterns in shapes that previously could not be detected automatically.

In comparison to the rigid detection method of Chapter 2, we extended the capabilities of the graph-based method (Figure 3.11, Figure 3.13). As shown in the examples we are always able to detect more instances than the improved implementation (Bokeloh et al. 2009) of our rigid method, while our deformable detection method is still able to detect rigid symmetries.

The most important disadvantage of our approach are the difficult to adjust parameters for the matching tolerances. Although we ease that a bit with the joint probabilities, they are still hard to find. Too small tolerances result in no matches at all, to large ones to false positives.

This issue has its origin in the question what is still symmetric and what is too deformed to be recognized as the same part. A way to overcome this issue, is to learn the answer from the data. In an attempt to do this, we present in the next chapter the concept of subspace symmetries, that reduces the variety of repeated parts that can be found to those that fit into low dimensional subspaces.

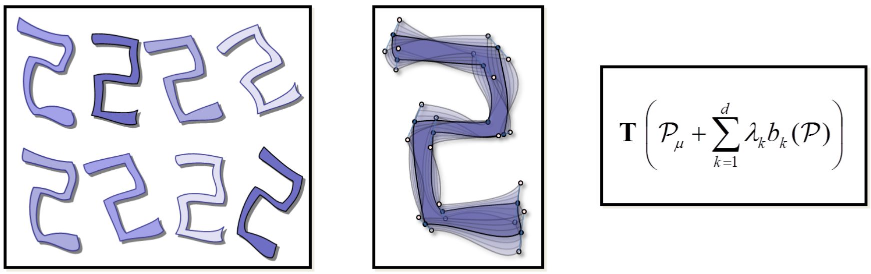
4
Shape Analysis with Subspace Symmetries

Our previous techniques identify parts that relate to each other by rigid mappings and relaxed intrinsic isometries. The approach we present now takes the next step and establishes a notion of partial symmetries for more general deformations: We introduce subspace symmetries whereby we characterize similarity by requiring the set of symmetric parts to form a low dimensional shape space. Our algorithm discovers subspace symmetries based on detecting linearly correlated correspondences among graphs of invariant features.

4.1 Introduction

To recap: Given an object $\mathcal{S}$, our goal is to look for a piece of geometry $\mathcal{U} \subseteq \mathcal{S}$ and a collection $\mathcal{F}$ of associated mapping functions $\mathbf{f}^{(i)} : \mathcal{U} \rightarrow \mathbb{R}^3$ that respectively create instantiations $\mathbf{f}^{(i)}(\mathcal{U})$, thereby matching the original geometry $\mathcal{S}$ approximately. Most efforts restrict the mapping functions to families of reflections, rigid mappings, and similarity transforms as shown in Chapter 2. Many real-world objects such as different windows of a building or ornamental structures in man-made sculptures, however, exhibit structural redundancies that cannot be captured by such constrained mappings.

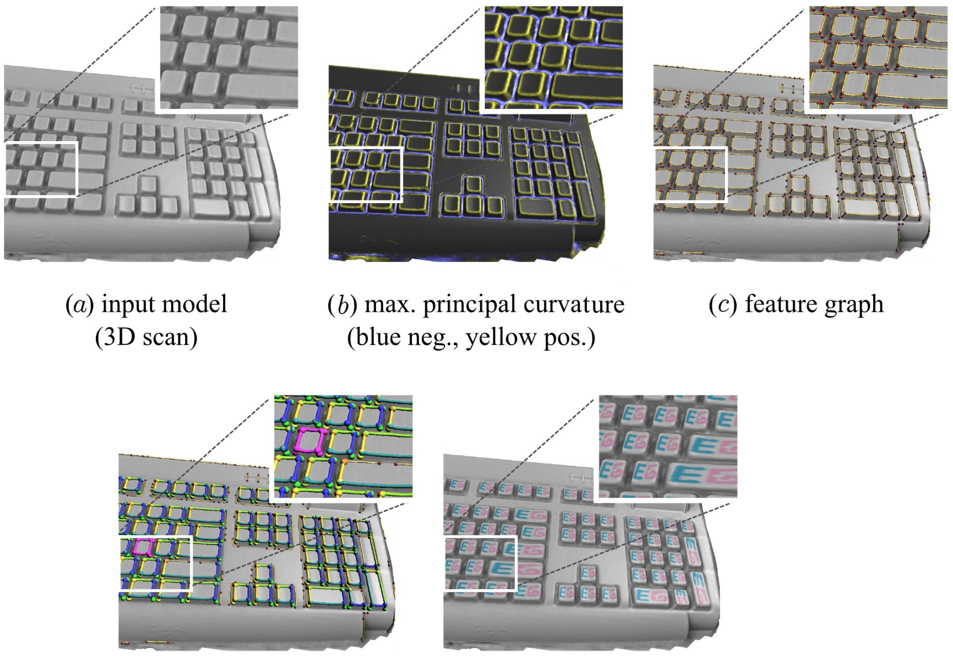
A central challenge in generalizing the notion of symmetry is to decide on the allowable space $\mathcal{F}$ of admissible transformations $\mathbf{f}^{(i)}$: While too much flexibility using numerous parameters lead to overfitting and spurious matches, an overly restrictive mapping fails to compactly capture redundancy present in the input. Our key idea is to not look at a prescribed set of admissible transformations independent of the input but first look for more general mappings supported by the input data. Then, we learn the space of useful variations by analyzing the ensemble of detected mapping functions, thus building a model for capturing *variations* exhibited by the input geometry, Figure 4.1.


$\mathbf{T}\left(\mathcal{P}_\mu + \sum_{k=1}^{d} \lambda_k b_k(\mathcal{P})\right)$

(a)Input shapes (b) Alligend shapes (c) Subspace encoding

Figure 4.1: *Given initial feature correspondences, we align the input shapes (a) with a transformation and move them by their centroid (b). (c) Now we can encode the aligned shapes (using* $\mathbf{T}$*) as mean shape* $\mathcal{P}_\mu$ *and its variations, the subspace.*

We propose a formal model based on this idea, introducing the notion of *subspace symmetries.* We constrain the mapping functions $\mathbf{f}^{(i)}$ to lie within a low dimensional affine subspace of all possible mappings. Thus, although each instance is described by a small amount of data, i.e., the low dimensional coordinate vector within the subspace, we still allow large variations for the individual mapping functions. In spirit of classical principal component analysis (PCA), the existence of a low dimensional structure within a high dimensional shape space serves as a validation criterion to characterize corresponding geometry. Our problem setting, however, is different from traditional PCA: We have to search for the useful correspondences, which are unknown, in the raw input to help reveal such a subspace structure. This is a very challenging search problem, and our work presents a first attempt for a practical solution.



(*a*) input model (3D scan) (*b*) max. principal curvature (blue neg., yellow pos.) (*c*) feature graph

(*d*) graph matches (corr. color coded, ref. pink) (*e*) dense correspondences (texture shows corr.)

Figure 4.2: *Finding subspace symmetries. We use features to find initial guesses for the correspondences that span the subspace. For an input model (a), we identify crease lines using extremal principal curvatures (b), and extract graphs of such crease lines (c). Subgraphs are then matched and refined using a learned subspace model to establish initial sparse correspondences (d). Finally, dense correspondences are estimated using a regularized subspace fitting technique (e). In comparison to Chapter 3, our subspace based method is able to detect instances with larger deformations and to recover broken local graphs, e.g. see closeup in (c).*

It acts in three steps (see Figure 4.2): The algorithm first searches for invariant patterns of features (crease lines), assuming that such feature constellations are preserved under the mapping functions. Once initialized with the sparse matches, the algorithm uses a variational regularizer to extend the mapping to dense correspondences and build the final subspace. We can then use this knowledge to refine further search results. Obviously, our approach is limited by the design and choice of the feature detection scheme. Nevertheless, to the best of our knowledge, this is the first algorithm for automatic detection of subspace symmetries. To overcome the restrictions to some extent, we also develop a semi-supervised variant of the algorithm that allows the user to bootstrap the subspace learning by providing a few hints regarding suitable features.

We tested our algorithm on a set of synthetic benchmark scenes as well as real world 3D scanner data in order to study its performance. For scenes with clear feature structure (such as Figure 4.2 and 4.11), we detect the dominant salient symmetries fully automatically. For challenging scenes, a small amount of user input often resolves feature matching ambiguities. Nevertheless, the requirement of clearly detectable invariant features remains as a restriction. As we will show, some models cannot be handled under this assumption.

4.2 Related Work

Shape analysis: Symmetry detection is widely used for pattern detection and regularity analysis in images and in 3D geometry. An object is said to be symmetric if it is (partially) invariant under the action of allowable symmetry transforms. A common approach is to identify a set of candidate transforms derived using potential correspondences, and map the correspondences to a space of transforms. As discussed already, this remodels the global problem of symmetry detection to local identification of clusters in the transformation space (Loy and Eklundh 2006; Mitra et al. 2006; Podolak et al. 2006). The approach has been extended to detect (commutative) Euclidean regularity (Pauly et al. 2008) and isometric regularity (Mitra et al. 2010) in 3D geometry. Such techniques are designed to handle transformation families that are characterized by a few parameters, e.g., translation, rotation, uniform scaling. However, generalizing the concept to handle other transformations involving many more parameters is challenging due to the ambiguity in the mappings, and difficulty in identifying good set of potential correspondences. Further, it is difficult to extend such techniques to learn data dependent allowable variation modes as discussed on the example of our relaxed intrinsic symmetry detection method in the last chapter.

The discussed enumeration based methods including geometric hashing (Gal and Cohen-Or 2006), robust auto-alignment (Simari et al. 2006), spherical harmonics analysis (Martinet et al. 2006), primitive fitting (Schnabel et al. 2008) are also inapplicable given the high dimensionality of the non-rigid transformation space.

Global Shape Registration: In shape retrieval, topological matching techniques have been used to recognize semantically similar shapes, e.g., (Hilaga et al. 2001). Zhang et al. (Zhang et al. 2008) propose a global shape matching framework based on comparing graphs of extremity features and evaluating the induced deformation of an assignment in order to match shapes such as humans or animals. The use of an elastic deformation model, however, limits the variability of models that can be handled. This problem has been partially addressed in (Au et al. 2010), where objects are reduced to skeletons and graph matching, and multi-dimensional scaling is employed to find similar skeletons. A similar idea has also been explored for global registration of animation sequences of 3D scans of deformable objects (Zheng et al. 2010). Existing methods for densely matching significantly dissimilar objects mostly assume a smooth mapping function such as thin plate splines (Allen et al. 2003), or work with restricted statistical models (Hasler et al. 2009), and require manual initialization of each match.

Dimension reduction: Projections on low dimensional affine (or nonlinear (Schölkopf et al. 1998)) subspaces have been used in a large number of computer vision and graphics applications. Eigenfaces (Kirby and Sirovich 1990; Turk and Pentland 1991) use low dimensional spaces to model photographs of human faces. The method was extended to handle geometric data by Blanz and Vetter (Blanz and Vetter 1999) in their highly influential work to construct a PCA space of faces from a collection of 3D range scans of images registered using optical flow. Allen et al. (Allen et al. 2003) extend the method to match different human body shapes using local optimization guided by manually annotated markers. Establishing dense correspondence allows the use of statistical learning techniques to describe spaces of plausible shapes, poses, and dynamics (Anguelov et al. 2005; Sumner et al. 2005; Hasler et al. 2009), and also for specific families of objects such as the shape of car bodies (Kkai et al. 2007). We explore affine 3D shape spaces with correspondences for symmetry detection – a direction that has been unexplored by previous methods and no global unsupervised or semi-supervised partial matching method has been provided in the cited work.

4.3 Subspace Symmetries

In this section, we introduce the notion of *subspace symmetries.* Our goal is to produce an *output* model comprising a set of shapes in correspondence such that they form an affine shape space. The key challenge is to simultaneously estimate shape spaces and their associated correspondences.

Affine shape spaces: Let a shape in form of a mesh $\mathcal{S}_i := (V^{(i)}, E)$ be represented as a set of vertices $V^{(i)} = \{\mathbf{v}_1^{(i)}, \ldots, \mathbf{v}_n^{(i)}\}$ and the connectivity structure encoded by a set of edges E.

Now let a set of shapes $\mathcal{S} := \{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ be in correspondence with each other sharing the same edges E. Thus, each shape $\mathcal{S}_i$ can be considered a point in a $3n$-dimensional shape space, i.e., $\mathcal{S}_i \in \mathbb{R}^{3n}$. Assume that the respective points of $V^{(i)}$ across shapes are in correspondence.

The set $\mathcal{S}$ is said to be spanned by independent *basis* shapes $\mathcal{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ and a mean shape $\mathbf{b}_0$ with $\mathbf{b}_i \in \mathbb{R}^{3n}$ sharing the same edge connectivity E, if and only if, each shape $\mathcal{S}_i$ can be *uniquely* expressed as:

$$\mathcal{S}_i = \mathbf{T}_i \left(\mathbf{b}_0 + \sum_{k=1}^{d} \lambda_k^{(i)} \mathbf{b}_k \right), \tag{4.1}$$

with $\lambda_k^{(i)}$ being scalar coefficients and addition referring to vector addition of respective elements of the vertex sets. Since such a linear space does not represent rotations well, we additionally store a rigid transformation $\mathbf{T}_i$ for each instance $\mathcal{S}_i$. Thus, shape S_i can be compactly encoded as $\{\lambda_1^{(i)}, \dots, \lambda_d^{(i)}\}$ and its rigid placement $\mathbf{T}_i$ in the scene. Given an example set $\mathcal{S}$, we can compute a model according to Equation 4.1: Following ordinary Procrustes analysis (Dryden and Mardia 1998), we first translate all shapes to align their centroids. Afterwards, we compute the relative rotation of the shapes $\{\mathcal{S}_2, \dots, \mathcal{S}_k\}$ to fit $\mathcal{S}_1$, which we pick as reference coordinate frame. The rotations are obtained by first estimating an optimal linear transform using standard least-squares, and afterwards projecting to $SO(3)$ by a polar decomposition that expresses the linear map as rotation and sheering component (which we omit). For this aligned set $\{\hat{V}^{(i)}\}$, we then perform a PCA: The mean vector of $\{\hat{V}^{(i)}\}$ yields $\mathbf{b}_0$, while covariance analysis of the mean centered vertex sets $\{\hat{V}^{(i)} - \mathbf{b}_0\}$ produces an orthogonal basis set $\{\mathbf{b}_1, ..., \mathbf{b}_d\}$, which spans the affine subspace. Further, the respective covariance values $\sigma_1, ..., \sigma_d$ along the principal directions encode the likelihood of variations by a positive definite quadratic form. In order to obtain a compact model, we remove insignificant principal components. We use the ratio to the largest eigenvalue as cut-off criterion (10% in our examples).

Correspondences: In the following, we use $\mathbf{f}_{i \to j}$ to denote the function that maps points $\mathbf{v}_k^{(i)}$ to their corresponding points $\mathbf{v}_k^{(j)}$ for $k \in \{1, \dots, n\}$. It is easy to see that these functions, after factoring out the rigid components, form an affine subspace of dimension d for any fixed i as well; correspondence functions could be seen as an alternative parametrization of the space. This also holds for correspondences from the mean shape $\mathbf{b}_0$ to models in $\mathcal{S}_i$.

Fitting the model to data: Once we know an affine model of the subspace symmetries, we can robustly perform model completion from imperfect data. Thus given a noisy and incomplete shape $\mathbf{s}$, we project the shape to the base space $\mathcal{B}$ by optimizing for coefficients $\{\beta_1, \dots, \beta_d\}$ that best represent $\mathbf{s}$ in the least squares sense. More specifically, our goal is to solve the optimization:

$$\min_{T, \{\beta_k\}} d\left(\mathbf{s}, \mathbf{T}\left(\mathbf{b}_0 + \sum_k \beta_k \mathbf{b}_k\right)\right), \qquad (4.2)$$

where, T denotes a rigid transform, and $d(\mathbf{a}, \mathbf{b})$ is a suitable (squared) distance measure between two shapes $\mathbf{a}$ and $\mathbf{b}$. We use the sum of squared

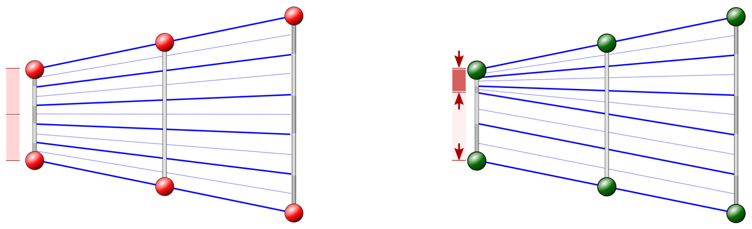


Figure 4.3: *Affine subspace correspondences are not always unique. The three flat shapes form a 1-dimensional shape space, but the two examples show two different solutions to the correspondence problem.*

point-to-plane distances (with truncation for far away points to improve outlier robustness) as our distance measure. We solve this optimization problem by iteratively computing the best rigid match to closest model points, using iterated closest point (ICP), and projecting the corresponding points into the PCA space. Later, we describe how to use feature based matching to initialize the optimization.

Invariance and regularization: In general, the restriction that the set of symmetry instances have to form an affine shape space of low dimension is not sufficient to uniquely establish correspondences across these shapes. For example, for corresponding planar regions in a shape we can find multiple different correspondence functions that form an affine subspace (see Figure 4.3). We therefore employ an additional (weak) regularizer: We minimize the spatial second derivatives of the correspondence functions $\mathbf{f}_{i \to j}$. This means, among all ambiguous solutions, we prefer the one with the least spatial bending of the correspondences functions as measured using a "thin plate spline" regularizer.

4.4 Extracting Subspace Symmetries

In this section, we discuss how to identify subspace models from input geometry that are used to find symmetric parts. The main challenge is to find correspondences that actually span a low dimensional affine shape space. Without any a-priori knowledge, this is challenging due to the combinatorial nature of the problem. A simple brute-force search would require computation time exponential in the number of correspondences involved, see Figure 4.4. We therefore propose an algorithm that is based on the additional assumption of the availability of *invariant* features: We assume that corresponding pieces of geometry show features that are invariant under the space of mappings $\mathbf{f}_{i\to j}$. On the other hand, matching features do not necessarily imply right correspondence.

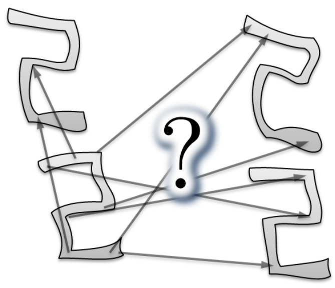


Figure 4.4: *Without initial correspondences, we cannot reasonable align the shapes or compute the subspace. We solve this using initial correspondences of symmetric features.*

We detect an arrangement of such features using the approach shown in the last chapter and use them to initialize a subspace search algorithm that establishes dense subspace correspondences (Section 4.4.2). Although this fully automatic symmetry detection approach works well on clean models, it may fail on challenging cases when the input is severely corrupted with noise and has large missing parts. Such failures arise in the initialization stage as our assumption on invariant features breaks down. In such cases, we propose a semi-supervised extraction strategy (Section 4.4.3) to allow the user to annotate a few training correspondences to seed the search for subspace variations.

Note that our feature matching strategy is not the only possibility, but other variations are conceivable, leading to similar or improved results. The focus of this chapter, however, is not on feature detection. Instead our goal is to detect subspace symmetries (mostly) automatically, but in a practically feasible way.

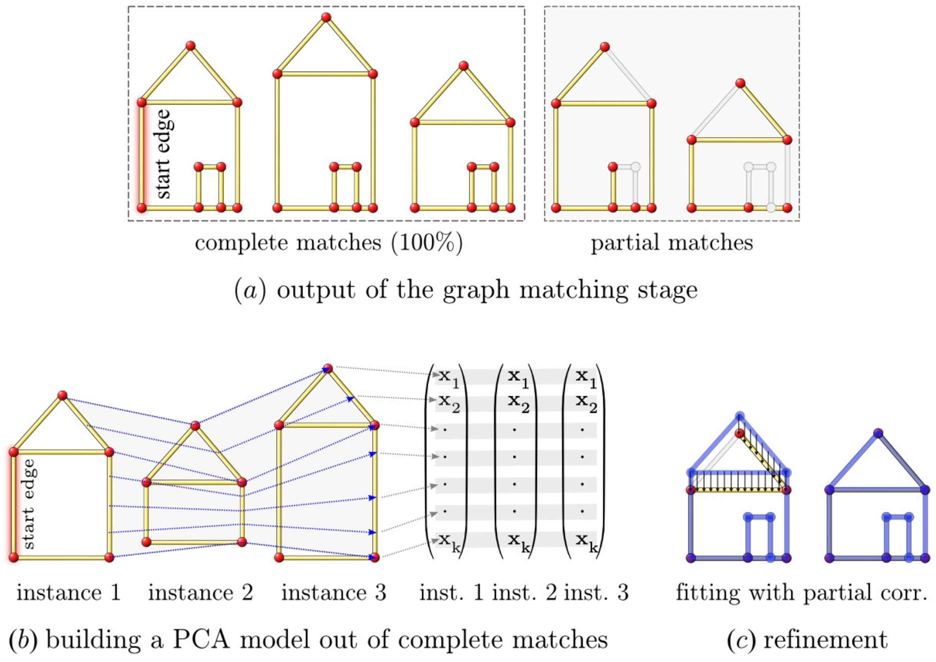


Figure 4.5: *Building and using a discrete subspace model. (a) Complete and incomplete graph matches; (b) the algorithm builds a model of the complete matches by parametrizing edges by edge length and mapping corresponding points to vectors that are used in a PCA analysis; (c) partial matches are refined and refitted using the learned PCA space.*

4.4.1 Graph Matching for Subspace Symmetries

We now show how the curvature line feature extraction of Chapter 3 is integrated in our method to identify subspace symmetries, see Figure 4.2). As explained, we extract a graph of features that is resilient to moderate deformations. The graph extraction is motivated by the observation that relationships across feature points and their connecting feature lines are better preserved under deformations, as compared to absolute geometry and thus suitable or subspace symmetries. We use crease lines of high curvature that have been demonstrated to capture shape characteristics (Ohtake et al. 2004), as described in detail in Section 3.4.

Now, we recap an overview of the graph creation and matching algorithm of Chapter 3 in the context of subspace matching.

Given the input model (Figure 4.2a), we estimate the principal curvature values and directions at every point of the model (Figure 4.2b). We then threshold the resulting scalar field keeping only points with large absolute maximum curvature. We place feature points at each line end and at each intersection. We then construct the feature graph with the feature points as nodes and the intermediate line segments acting as edges (Figure 4.2c).

We now use the computed graph to search for candidate matches. Based on our assumption that partial similarity between adjacent feature lines remains invariant under moderate deformations, detecting repetition patterns amounts to solving a partial graph matching problem.

We apply the graph matching method (Section 3.5) to search for candidate matches of subspace symmetries. But instead of returning complete matches only, we also return incomplete partial matched patterns (Figure 4.5a).

After this initialization step, we have a list of partial subgraph matches. Each match starts from a source graph containing the start edge, and maps to different target graphs, only partially overlapping in the source domain. For further processing, we only handle instances with sufficient overlap, thus describing the same instance. Therefore, we delete all matches without substantial overlap (typically, 60% of the candidate matches) with the largest found match. The remaining partial graph matches are used in further processing (Figure 4.5a).

We also determine all the complete matches and use them to initialize the subspace model. The partial matches are used as candidates to be matched to the subspace model. For increased robustness, we iterate the whole graph matching pipeline multiple times with random seeds in an outer loop. We keep the solution that maximizes the product of the number of instances that are in complete correspondence and the number of edges involved.

Building a discrete subspace model: We now compute a subspace approximation of the candidate salient line patterns, which are likely to correspond to actual subspace symmetric geometry. First, we establish correspondence across the line segments using a simple arclength parametrization (Figure 4.5b), normalized to overall unit length. Points at the same distance from the start vertex are then set to be corresponding. We sample each edge uniformly and form a long vector of corresponding points. We compute a subspace representation of the form of Equation 4.1 using PCA, with rigid transforms factored out, as described in Section 4.3. We keep those eigenmodes with eigenvalues of at least 10% of the (unsigned) magnitude of the largest.

Discrete refinement: Using the learned subspace, we now refine the remaining matches by considering the previously unconsidered partial matches. The local geometry of the candidates are validated in the subspace model of the feature lines (Figure 4.5c) by minimizing Equation 4.2, as discussed in Section 4.3. For increased robustness, we only use the previously extracted points of high curvature as target shape for the alignment since the feature lines cannot map to flat regions. After matching, we measure the Mahalanobis distance to the PCA model, and discard matches with a distance larger than three times the standard deviation. Note that our model space is significantly low rank in proportion to the high dimensional embedding shape space. Hence, a slight amount of random noise can lead to, with high probability, unstable Mahalanobis distance. Therefore, we assume a small uniform covariance of $\sigma^2\mathbf{I}$ in all directions, and add this to the covariance obtained from the PCA analysis. σ corresponds to the noise level in the data (including small effects not modeled by our subspace model).

Note that the refinement step is critical to a viable solution to the subspace symmetry detection problem. Importantly, the subspace symmetries can be initialized with a small amount of data. Subsequently, we use the model to identify and verify candidate matches arising out of partial and ambiguous data. Figure 4.2d shows the final result of the discrete matching after discrete refinement, while Figure 4.6 shows how PCA coordinates can be learned and how a partial example match is found.

4.4.2 Dense Subspace Correspondences

Corresponding feature line skeletons that form a low-dimensional subspace are good indicators that the dense geometry points enclosed by the cells of these feature graph also form a low dimensional subspace. To test such candidates,

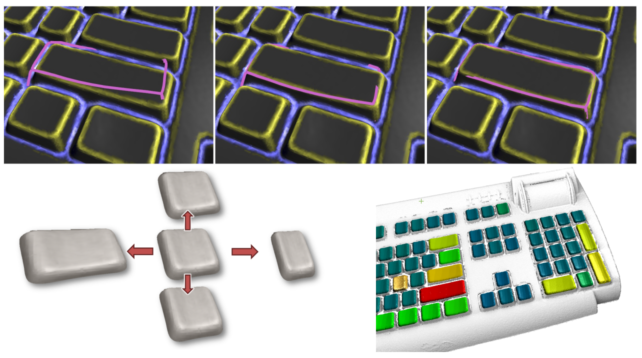


Figure 4.6: *Subspace assumption helps in refining matches. (Bottom-left) The two principal eigenmodes of the "key" element, (top) iterative fitting, alternating between rigid alignment and subspace coordinates, (bottom-right) rainbow color coded dominant eigenmode for signed subspace coordinates for all detected matches, blue being minimum.*

we bring the dense points into correspondence and then again perform PCA while factoring out rigid alignment.

In order to extend the discrete matches to dense correspondences, we use a regularized deformable ICP as introduced in Section 3.6. We regularize our mappings f using a thin-plate-spline energy (as motivated in Section 4.3), seeding the optimization using the computed discrete matches as known boundary condition correspondences.

We perform this matching for each of the found (discrete) symmetries, thus resulting in dense correspondences encoded by a number of matching functions $\mathbf{f}_i := \mathbf{f}_{1\to i}$, where we use index one to denote our start instance, i.e., the initial location of the start edge.

Finally, we compute a space $\mathcal{S} = \{\mathcal{S}_1, \mathbf{f}_2(\mathcal{S}_1), \ldots, \mathbf{f}_k(\mathcal{S}_1)\}$ to compute the final subspace model using rigid alignment followed by a PCA analysis.

4.4.3 Semi-Supervised Algorithm

When the input objects have significant variations or are highly corrupted by noise, our initial discrete feature graph correspondences can be unstable, and fail to suitably seed the symmetry subspace search. In such cases, we provide an optional mechanism to train the feature matching model in a semi-supervised fashion. Although other feature detectors may produce improved performance, we believe that in certain cases user inputs are unavoidable to provide unambiguous leads for a reliable creation of initial subspace candidates, which can then be automatically refined and other instances learned.

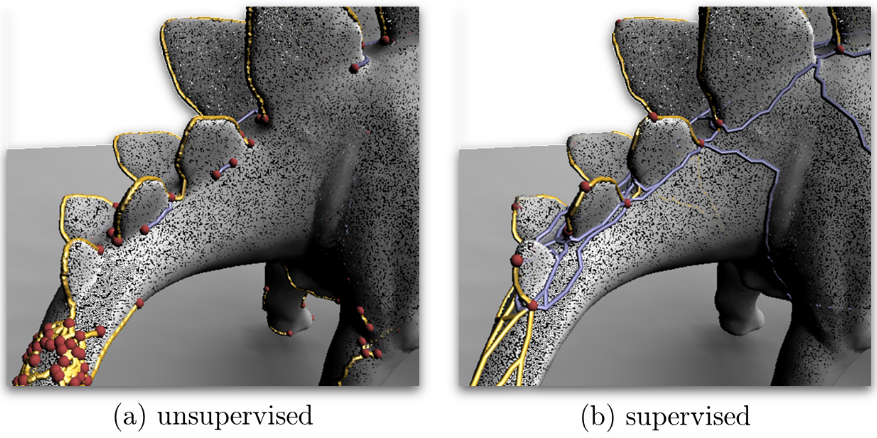


Figure 4.7: *Comparison of the semi-supervised feature lines. (a) Result of our curvature based feature line detection method.(b) Result of the proposed semi-supervised approach that allows the user to click a few points of interest.*

We ask the user to click on a few feature points that should be in correspondence. The algorithm computes a descriptor of the local geometry for each point using histograms of intrinsic distances to nearby samples within a neighborhood ball. We build a PCA model of the descriptors and add a regularizer $\sigma_{descr}\mathbf{I}$ to the covariance matrix to account for noise. The user interactively specifies σ_{descr} checking when false positives start appearing. Using the PCA model, we automatically detect all points on the model with descriptors that fall within the covariance of the model, using a maximum Mahalanobis distance of three sigma. Next, we connect the feature points using geodesic paths on the underlying surface. We simultaneously grow regions from all the feature points to compute intrinsic Voronoi regions, and join only those point-pairs that have a connecting edge in the dual triangulation.

A comparison of the automatically generated lines and the feature graph based on user interaction is shown in Figure 4.7. Please note that the interaction only affects the feature line creation step. Every subsequent step of the pipeline remains unchanged and unsupervised.

4.5 Applications

Instance replacement: The detected correspondence fields $\mathbf{f}_{i \to j}$ between all instances enable non-local edits, which are otherwise difficult to perform. Base geometric patches can now be easily textured, altered, and replaced, and the edits automatically propagated to all the symmetric instances. We use this tool to show the correspondence mapping across the instances.

Shape completion: Once a subspace model is discovered, we can robustly detect partial matches by verifying potential candidates against the subspace. This allows us to automatically repair incomplete geometry, which is common due to occlusion and scanning artifacts. We use the partial feature graph to initialize the matching and compute the least squares best fitting mapping function $\mathbf{f}$ by projecting the constraints into the subspace. Finally, we deform the base geometry using the established spatial mapping function $\mathbf{f}$ for shape completion.

Denoising: We use the established correspondences for scan denoising. This mode is particularly interesting since we can establish general, non-rigid mappings that previous techniques fail to detect. For denoising, we first compute the mapping functions from the mean shape $\mathbf{b}_0$ to all other instances as $\mathbf{f}_{mean \to i}$. Using the inverse mappings $\mathbf{f}^{-1}_{mean \to i}$ we then copy all the data points to the mean shape domain. Finally, we use a standard moving least squares reconstruction in the mean domain for denoising, and transfer back the results using the original maps $\mathbf{f}_{mean \to i}$.

4.6 Results and Discussion

We tested our algorithm on a variety of models, both synthetic and scanned. All the scanned model point clouds are reconstructed using Poisson surface reconstruction to generate approximately isotropic meshes as inputs.

Results of our algorithm are depicted in Figure 4.2, 4.8, 4.11, and 4.12. We visualize the discrete matches using the same color as the corresponding graph elements. In order to display the dense correspondence, we paint a texture on one of the instances and transfer it to the other instances using the computed deformation fields $\mathbf{f}_i$. Several examples contain self-symmetries, such as a chair that can be mirrored along its side and mapped back to itself. Our algorithm detects all (for synthetic) or most (for scanned examples) of these additional symmetries. However, for clarity of presentation we show only the global self-symmetry of the instances: We only visualize the match whose rigid component has the smallest deviation, in the Frobenius sense, from the identity map.

Synthetic test scenes: We first report experimental results on clean synthetic datasets, which are not corrupted with noise, and are complete. For such models, feature matches provides good seeds for the subspace search, and we reliably obtain perfect results, i.e., our algorithm finds all the non-rigid self-similarities that we expect to extract from the data (see Figure 4.8). As our algorithm works at the level of feature graphs, the matching results do not depend on the tessellation of the meshes, as long as the underlying geometry remains unchanged.

Real-world 3D scans: The problem quickly becomes challenging for scanned inputs, which usually contain various artifacts including ambiguity due to noise, missing data, and allowed non-rigid deformations. Our first dataset is a scanned PC keyboard (see Figure 4.2). This data set reveals a clear feature structure that is extracted by our automatic algorithm. As shown in Figure 4.6, we obtain a subspace model with two dominant directions of variations, which are sufficient to match all keys in the keyboard with high precision. Thanks to the initial Poission surface reconstruction, our algorithm is quite robust under increasing noise levels. Figure 4.17 shows the same scene with artificial Gaussian noise added to the original scan. Recognition becomes problematic only after the noise level starts blurring out the feature lines. Figure 4.11 shows the results for a LiDAR scan of a 14th century gothic church – the *Marktkirche* from the well-known scan repository provided by the IKG of Hannover University. Our algorithm detects the repeated windows of varying shape, and subsequently establishes dense correspondences across the instances.

Next, we use a scan of a small clay replica of a statue (Figure 4.12), which provides a challenging example due to variations in clay instances. Again we automatically discover most of the ornaments below the neck of the statue.

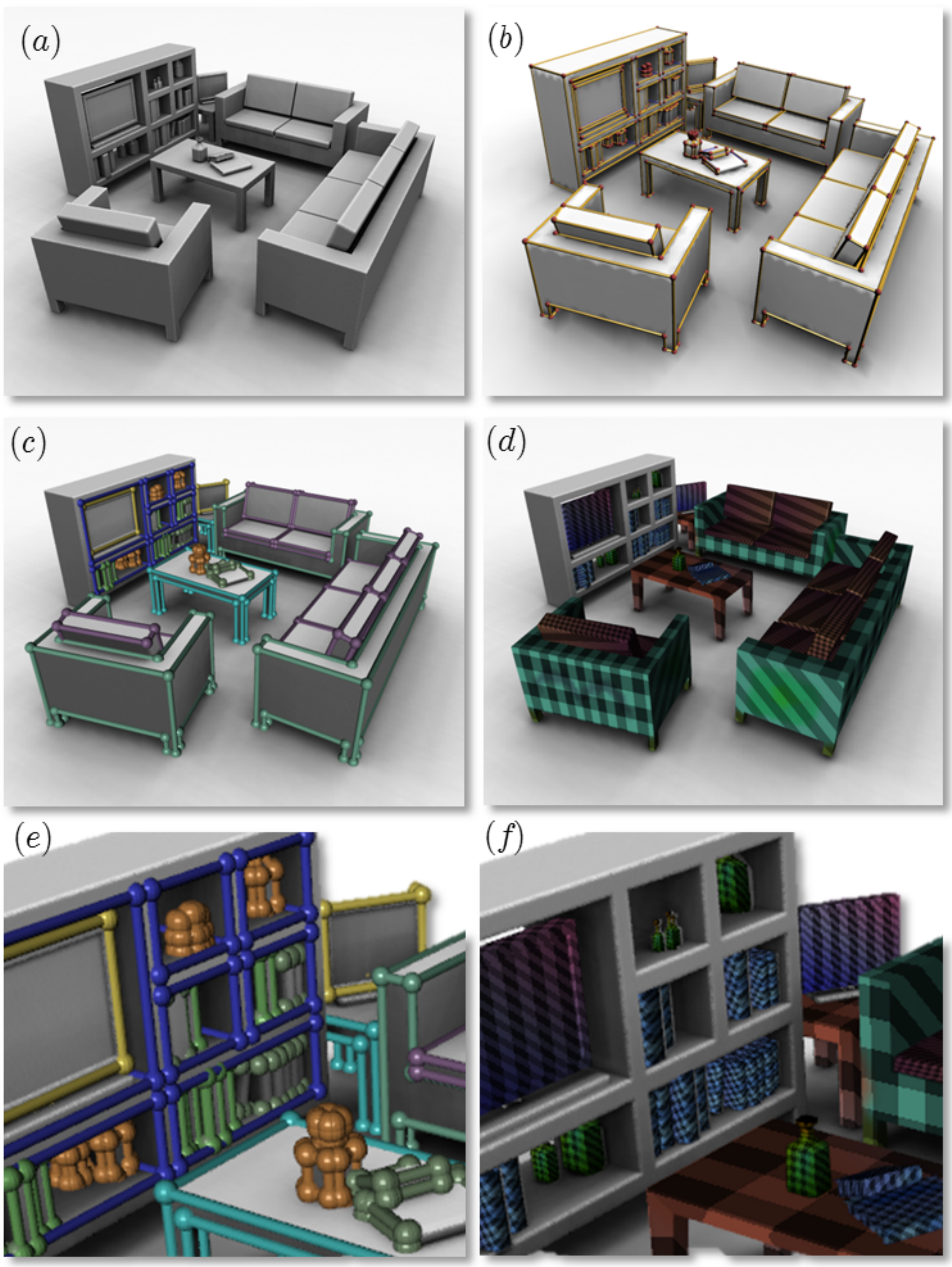


Figure 4.8: *Detected subspace symmetries a synthetic example. The computed dense correspondence are visualized using mapped textures. (a) input model (mesh), (b) feature graph, (c) discrete matches and (d) dense correspondences, with (e) and (f) showing closeups.*

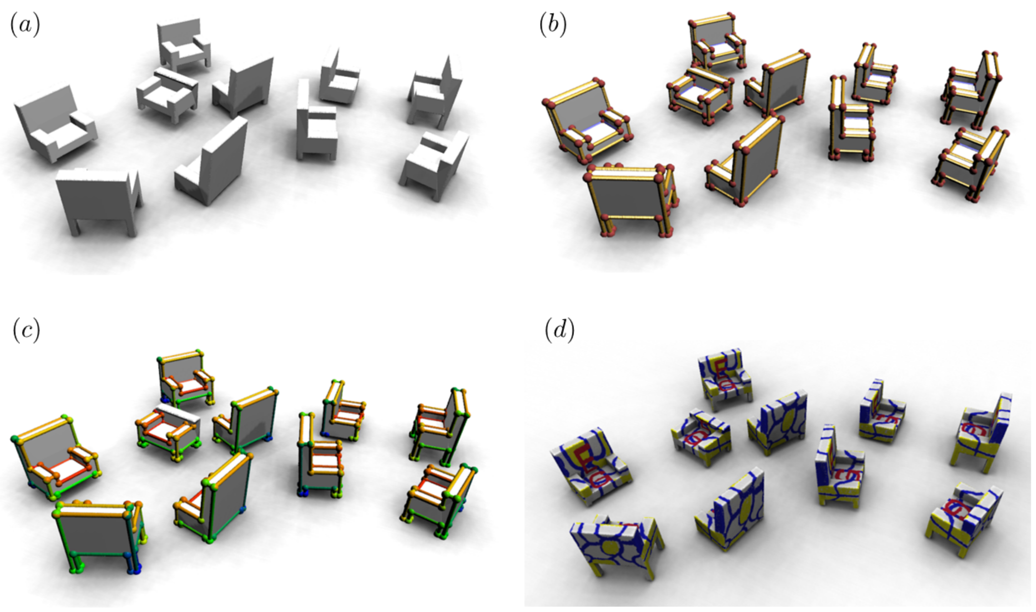


Figure 4.9: *Detected subspace symmetries on a synthetic example. The computed dense correspondence are visualized using mapped textures. (a) input model (sampled mesh), (b) feature graph, (c) discrete matches and (d) dense correspondences.*

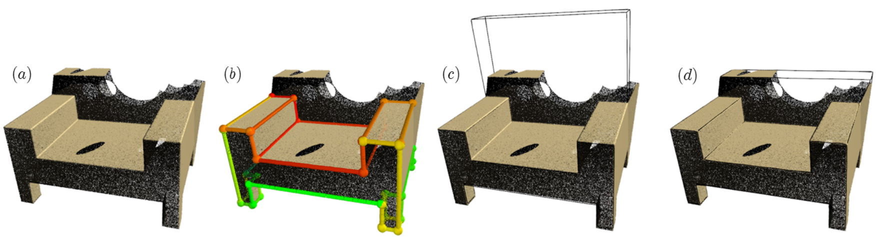
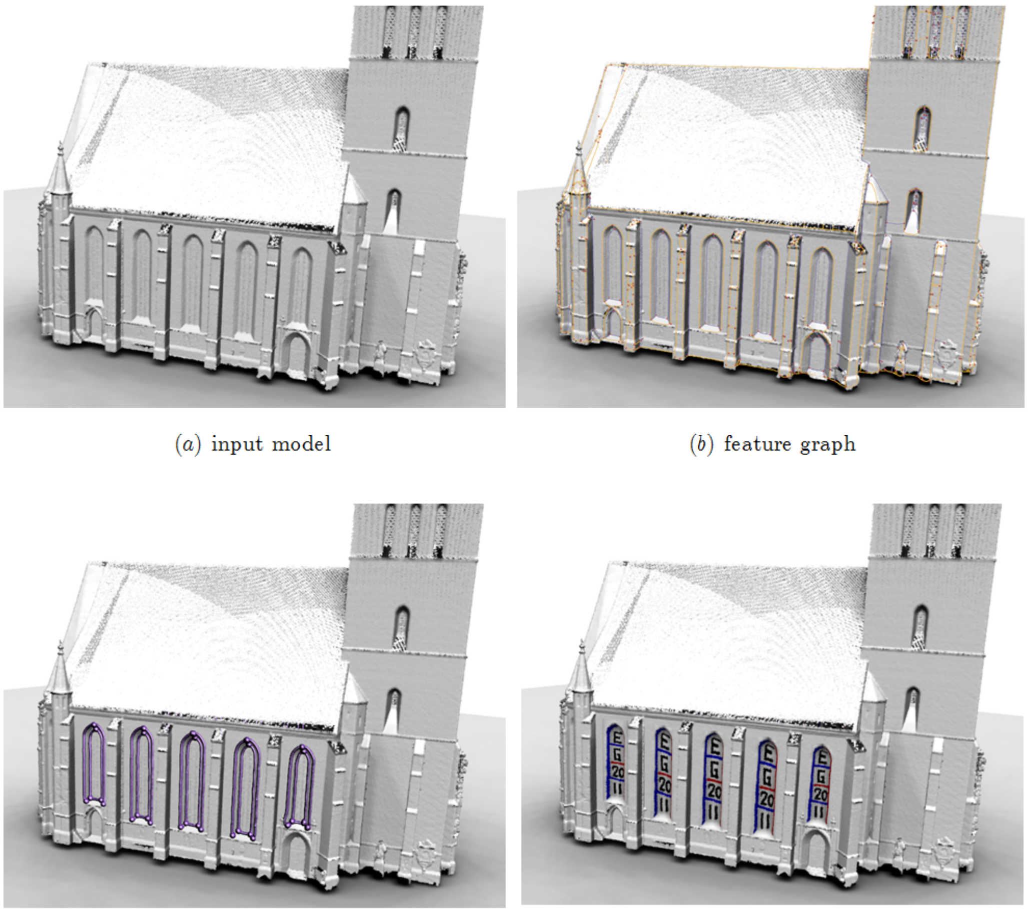


Figure 4.10: *Subspace assumption helps in verifying and refining matches. (a) Defect instance of the chair data set (random ellipsoids deleted), (b) we detected a partial discrete symmetry, (c) we initialize our PCA model with the partial match, (d) iterative fitting, alternating between rigid alignment and subspace coordinates converges to the correct solution.*



(*a*) input model (*b*) feature graph

(*c*) discrete matches (*d*) dense correspondences

Figure 4.11: *Automatic detection of subspace symmetry on a scanned point cloud data of a church ("Marktkirche" in Hannover).*

However, to resolve the *grid ambiguity*, we manually designate the “cut-out” of one instance (shown in purple). Otherwise, the algorithm detects larger instances with several pieces combined that are mapped in groups. Although correct, this creates a large, overlapping set of detected parts. A discrete group reduction algorithm (Mitra et al. 2006; Pauly et al. 2008) acting on the discrete permutation group of the detected feature correspondences could resolve this issue automatically. As grid detection is not the main focus of this work, we leave this for future work.

The automatic mode fails on the most complicated input example, as shown in Figure 4.13a, since we fail to get a good set of seeds for initial subspace construction. Therefore, we allow the user to click on example features (two different tips of the scales, two more examples at the bottom). Afterwards, one example graph (one scale) is selected. This information is sufficient to remove distracting features and recognize all the major scales on the back of the dinosaur. We obtain a subspace with overall four main directions. In particular, the first two are intuitively interpretable, encoding size and skewness of the scales (see Figure 4.13d). Subsequent dense correspondence establishment works well.

Shape completion and denoising: We demonstrate denoising and hole filing on the church data set (Figure‘4.15 and Figure 4.16). The algorithm detects a partial match and fills in geometry that is closest in a least-squares projection to the learned subspace. Similarly, we combine the geometry information across all the instances and perform a non-local, non-rigid denoising (Figure 4.15). This reproduces fine details and sharp edges better than in any single instance of the original scan. However, some subtle details that vary across the instances are lost by our algorithm when working with the available scan resolution (by looking very closely at the original data, one can guess that there are a different number of glass elements in large vs. small windows).

Timings: All the examples ran in the order of 5-10 minutes on a 2.5 GHz Core2Duo laptop with 8GB RAM with un-optimized code with the following breakup: each RANSAC step inner loop ran in about 5-30 seconds depending on graph complexity, line PCA building took around 30 seconds, and instance refinement order of few seconds, and finally, the most expensive dense cor-

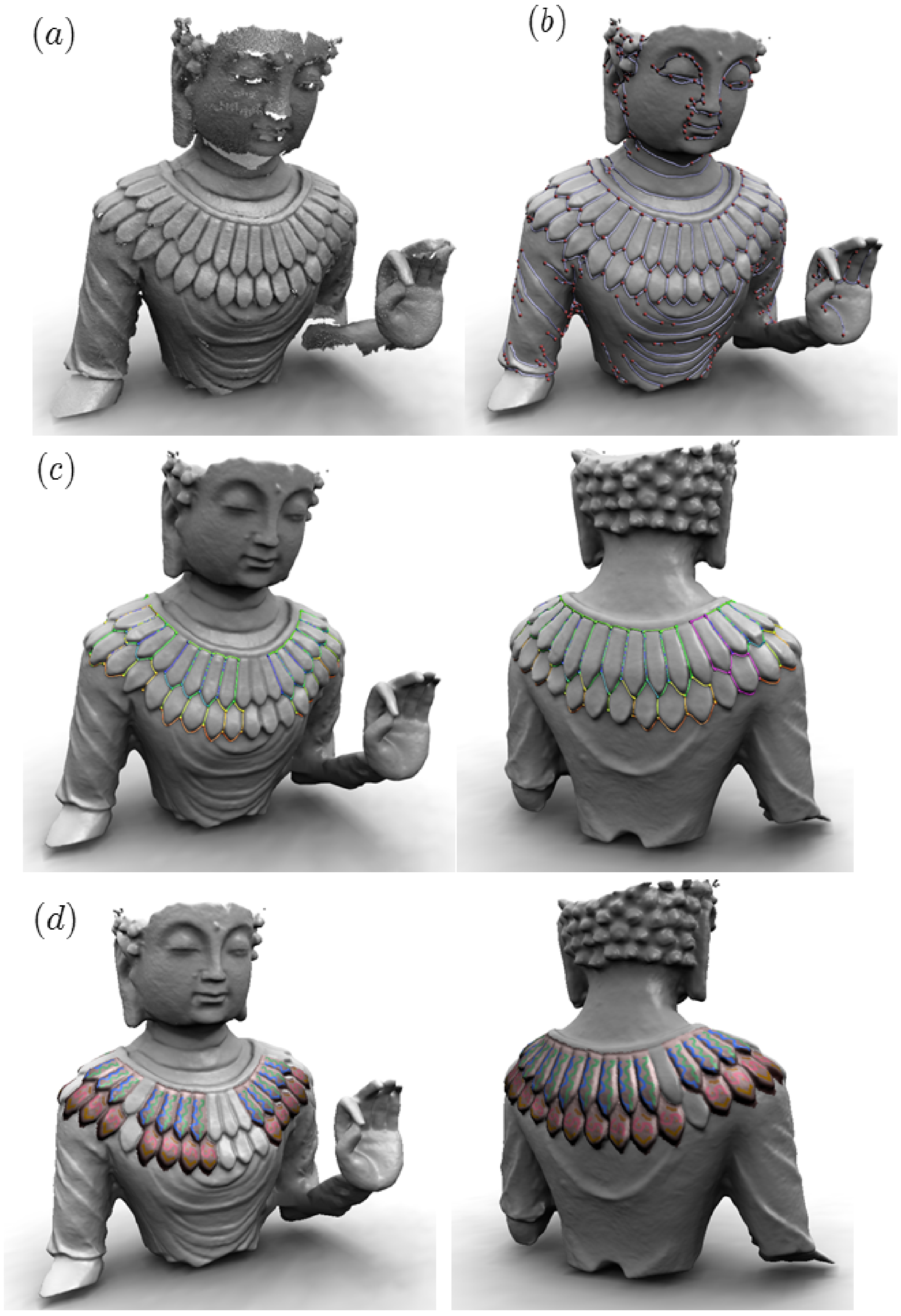


Figure 4.12: *Statue scan; matching is fully automatic. However, we chose one grid element manually to resolve grid ambiguities. (a) input data, (b) feature graph, (c) discrete matches: color encodes corresponding parts, purple = designated grid element and (d) dense correspondences.*

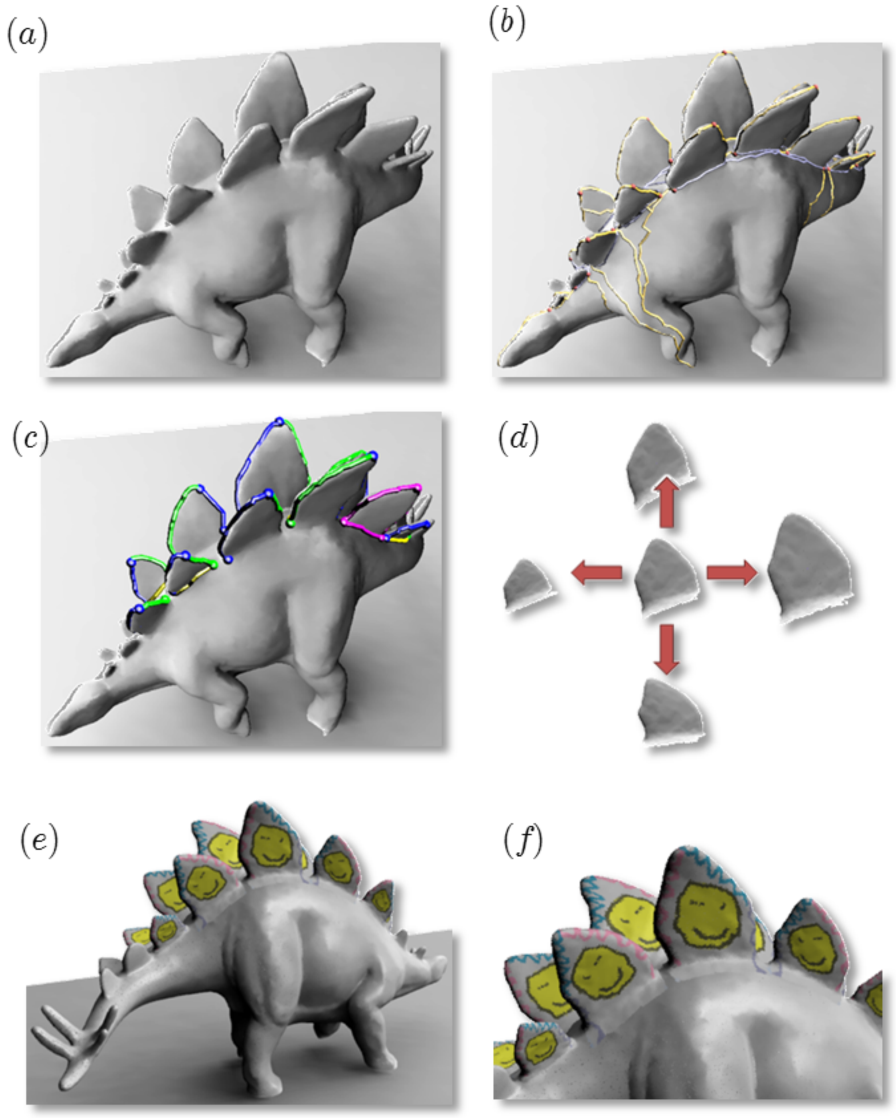


Figure 4.13: *Dinosaur model scan. The user specifies 4 example features and one subgraph(purple) to bootstrap the detection. (a) input model (Poisson mesh), (b) feature graph, (c) discrete matches, (d) Shapes along the two dominant eigenmodes of the plates, (e) dense correspondences and closeup (f).*

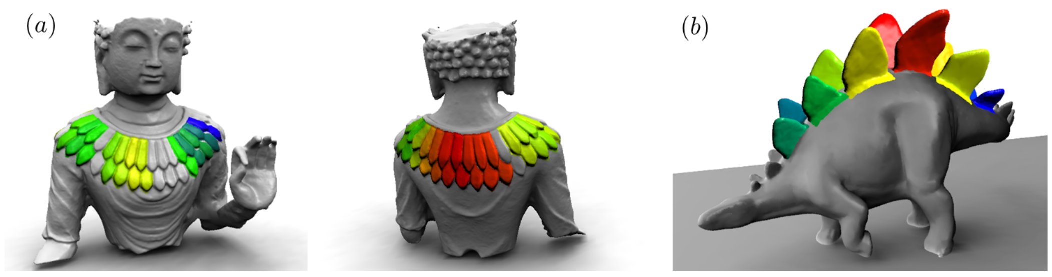


Figure 4.14: *Visualization of the normalized signed coefficients along top eigen-direction, using the rainbow map, blue being minimum.*

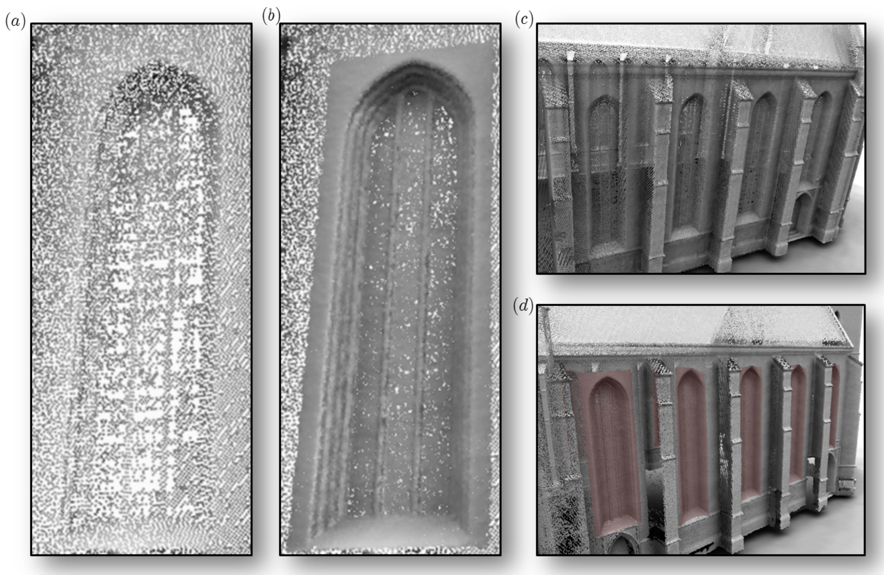


Figure 4.15: *Automatic non-local non-rigid denoising and super resolution: The deformed instances are overlaid and projected to the subspace for denoising and detail transfer.*

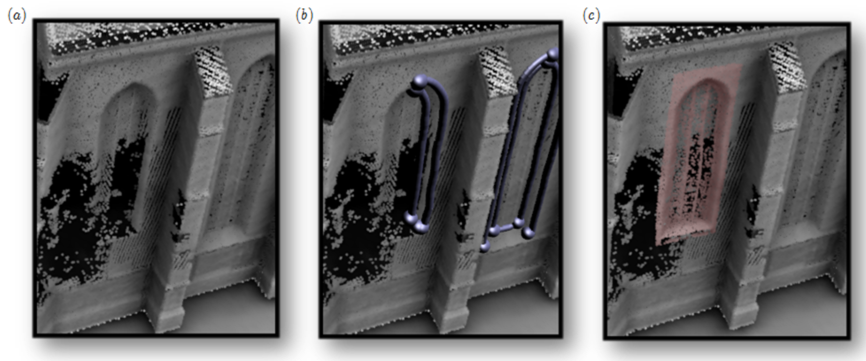


Figure 4.16: *Automatic non-local non-rigid hole filling: A partial match is identified and filled with matching geometry from the subspace.*

respondence ran in around 3-8 minutes depending on the complexity of the symmetries and the grid resolution of thin plate spline solver, with dense PCA taking less than a minute.

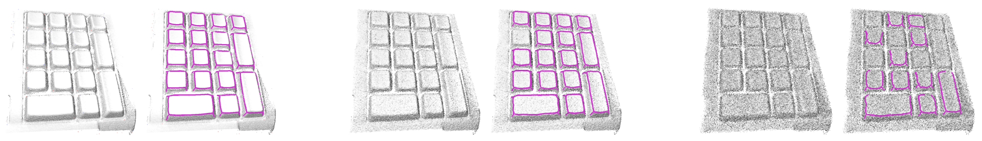


Figure 4.17: *(a)-(c) robustness to noise test: noise levels are the standard deviation of Gaussian additive noise as percentage of the largest side of an axis aligned bounding box of the whole scene.*

Limitations: The main limitation of our subspace symmetry analysis algorithm stems from the assumption that the chosen feature detector can detect enough nodes to seed a subspace model search. In presence of significant noise or large missing parts, this assumption breaks down for our choice of feature curves, thus forcing the algorithm to switch to a semi-supervised mode. Even then, our model does not capture all possible cases. For example, the structures on the wing of the "Gargoyle" model in Figure 4.18b do not lead to intersecting graph edges. The other line pattern on the rest of the model are also not clear enough to seed subspace models (even with some user interaction to improve the feature graph). Furthermore, the features guided matching in general has the drawback that it involves manual setting of parameter values to establish the graph.

The success of the approach is dependent on the richness of the line feature invariant subspace symmetry present in models. Although we found many man-made models to be a rich source of such symmetries, as shown in our experiments, there is a large class of models that can not be handled using our current feature choice. An example is the pig model in Figure 4.18a. While we can obtain good feature graphs at the ears, we cannot find such graphs for other symmetric parts e.g. the four legs. They are nearly perfect cylinders and offer no crest lines along them. In the next chapter, we will discuss a user guided method to compute symmetry correspondence mapping for this class of models.

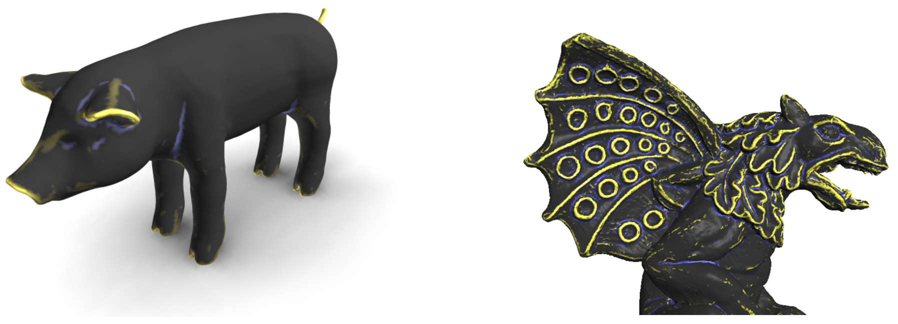

(a) pig model (b) "Gargoyl" data set

Figure 4.18: *Failure cases: (a) Our feature line graph creation relies on lines of high curvature that are rare in this model. Data set provided by (Giorgi et al. 2007). (b) The "Gargoyl" statue has clear crest lines, but we can not find intersecting graph edges in the repeated circle ornaments on the wings. Data set provided by the AIM@SHAPE repository (http://shapes.aimatshape.net).*

4.7 Summary

In this chapter, we introduced subspace symmetries to capture similarity in the surface geometry, which are related by non-rigid transformations that are not arbitrary but span a low-rank subspace. The resultant symmetry subspace then has a natural compact description, and effectively captures the variations of the underlying surface. We presented an algorithm to detect such symmetries, both automatically and also in a semi-supervised mode. The subspace symmetry detection algorithm was tested on various classes of inputs. Further, the extracted symmetry subspaces enable a range of interesting geometry processing tasks including non-local non-rigid denoising, model completion, simultaneous instance replacement, while factoring out the underlying subspace variations.

Further efforts are needed to explore alternate feature descriptors that can robustly initialize and seed a subspace model search. Using the crest lines only obscures some symmetries like in Figure 4.18a. A combination with other feature types, e.g. shape primitives of (Schnabel et al. 2007) can help here.

Capturing the model variations and the distributions of the embedding parameters can allow efficient generation of statistical variations in shape spaces, thus producing subtle data-driven variations in shape families. For example, we can sample and set the subspace coordinates of the identified parts e.g. for the scales on the back of the dinosaur (Figure 4.13) and modify the geometry accordingly.

This way we can manipulate the model staying in the shape space of parts or sample from the shape space of parts and create a valid new model with similar appearance. An obvious problem in this scenario is the interaction of the parts if one is manipulated and changes its boundaries. The operation can introduce gaps and discontinuities. In the next chapter, we address these issues and introduce an interactive shape manipulation framework based on subspace symmetry editing.

5
A Morphable Part Model

In the last chapter we presented an algorithm that automatically identifies sets of symmetric parts to form a low dimensional shape space. Using a feature based bootstrapping, we initialize a morphable model of a shape varying part to search for other parts belonging to this set. After the process is finished, we obtain morphable models of all found parts as a spin-off. Our key idea here is to employ this information to manipulate a model or to sample from its part shape spaces. We develop methods for the interaction between the parts while editing and discuss how to compute the part morphable models if the shown automatic method fails.

For now we assume we are given the part regions and dense correspondences for the input models. Our method then conducts two phases: An analysis phase to establish the morphable part models thereby learning inter-part relations and an interaction phase where editing and sampling is performed in real time.

The discrete aspect of how parts can be assembled is captured using a shape grammar in the analysis phase. We show how the parts and their interconnection rules are learned automatically from symmetries within a single object or from semantically corresponding parts across a larger set of example models.

In the interaction phase, we obtain an interactive yet intuitive shape deformation framework producing realistic deformations on classes of objects that are difficult to edit using existing deformation techniques. Unlike previous techniques, our method uses self-similarities from a single model or corresponding parts of collections as training input and allows the user to reassemble the identified parts in learned new configurations, thus exploiting both the discrete and continuous learned variations while ensuring appropriate boundary conditions across part boundaries.

5.1 Background

Developing simple yet expressive deformation models to facilitate interactive and intuitive manipulation of geometry remains one of the important research areas of geometric modeling, interaction, and animation. Typically, the user specifies desired positions for a few handle points on the input pose, and the goal is to automatically deform the rest of the input model into a plausible new pose. Although a large variety of deformation techniques exists, the methods primarily differ based on what geometric properties are preserved during deformation. Notable recent approaches include preserving local surface details (Sorkine et al. 2004), keeping local elements as-rigid-as-possible (Igarashi et al. 2005; Sorkine and Alexa 2007), achieving isometric deformations (Kilian et al. 2007; Winkler et al. 2010), or respecting global relations across object parts (Gal et al. 2009; Zheng et al. 2011).

Unfortunately, many shapes, especially organic ones, show up in significant shape and pose variations. Many such classes of shapes cannot be related by local-rigidity, isometry, or similar distance measures based on just preserving local differential properties. Instead, the range of possible deformations are what actually characterizes the respective objects, and cannot be specified a priori. This motivates a data driven approach to first *learn* the space of allowable deformations, and subsequently *restrict* deformations only to the learned deformation space.

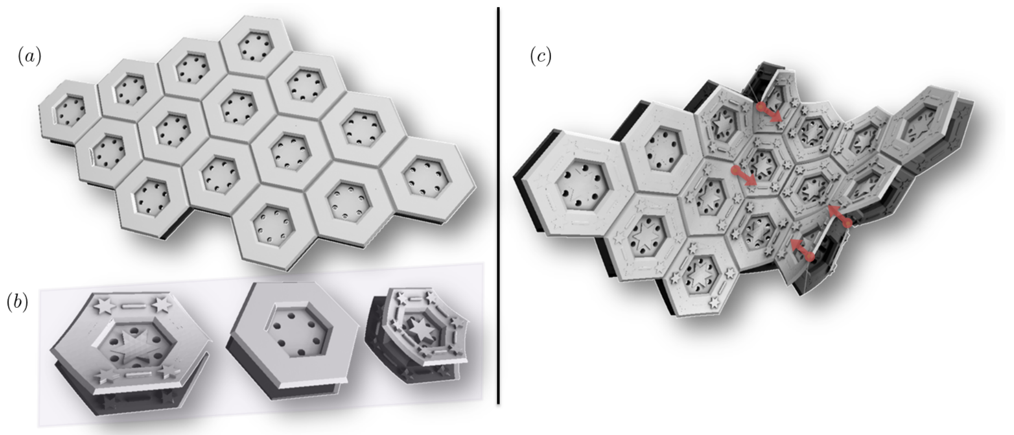


Figure 5.1: *An assembly of artificial parts to illustrate the morphable part based concept of our method: Three variants (b) of a part with different bending are learned from a three example models. In an assembly (a), parts react to neighboring tiles and morph according to the examples under user constraints (red) to best fit the learned model (c).*

Given a set of reference model poses, one possibility is to establish a global correspondence across the multiple poses, and then build a statistical model to capture the dominant variations using a set of extracted parameters. Such a model is commonly referred to as a *morphable model*. The most commonly used statistical model is linear principal component analysis (PCA), which computes a global Gaussian model of the vertex positions of the mesh. Please see the introduction to PCA in the last chapter, Section 4.3. In various applications, morphable models have produced impressive results (Blanz and Vetter 1999; Cootes et al. 2001; Hasler et al. 2009).

However, there are some limitations: A global Gaussian model directly captures the correlations of all model points. This leads to a combinatorial blow-up for objects with many independent degrees of freedom: For example, in a human body shape, a large number of combinations of poses of the individual body parts need to be observed in the training data so that a correct model can be learned. Furthermore, it is not possible to recombine parts because the analysis is based on global correspondences between all of the described shapes. For example, in Figure 5.11 deformations of the spider's body and legs are better described by separate morphable models. Nevertheless, the parts are mutually correlated, e.g., the spider's body cannot shrink without adjusting the size of the legs.

In this work, we propose a part-based morphable model to overcome these limitations. We capture variations of individual parts of an object along with the mutual dependencies across the parts, where both the part-variations and the inter-part dependencies are learned from given correspondences. We encode the variations as a graph where each node represents an object part, while the edges store the relations among the parts. Thus the continuous variability is captured by attributes at the nodes and the edges, while the discrete variability is stored as the graph connectivity. We semi-automatically learn such a *morphable part model* starting from an input set of training poses. The parts and their relations are combined using an elastic deformation model that connects different parts while ensuring seamless stitches and globally distributing deviations from the observation. Our model corresponds to a Gaussian Markov random field (MRF) rather than a global Gaussian model, which gives us the opportunity to describe part behavior and part interaction locally, and gives us a well defined interface to rearrange parts in different combinations. We reduce the resultant system to solving a large sparse Laplacian matrix and achieve interactive performance using a Schur complement decomposition.

We use our framework towards an intuitive deformation framework by restricting deformations to the learned morphable part space obtained by examining partial, deformable symmetries of the input shape e.g., using subspace symmetries of the last chapter or corresponding parts from a larger collection of example shapes. As a special case, partial symmetrization is achieved when corresponding parts are approximated by the average part from the respective shape space while maintaining the global structure of the object.

In another application, we support both discrete and continuous shape edits. Based on user annotations of a shape database, we first extract continuous and discrete rules to encode the space of deformations prescribed by the input poses. We then construct new graph variants that are compatible with the learned model and subsequently embed the graph in 3D using a least-squares

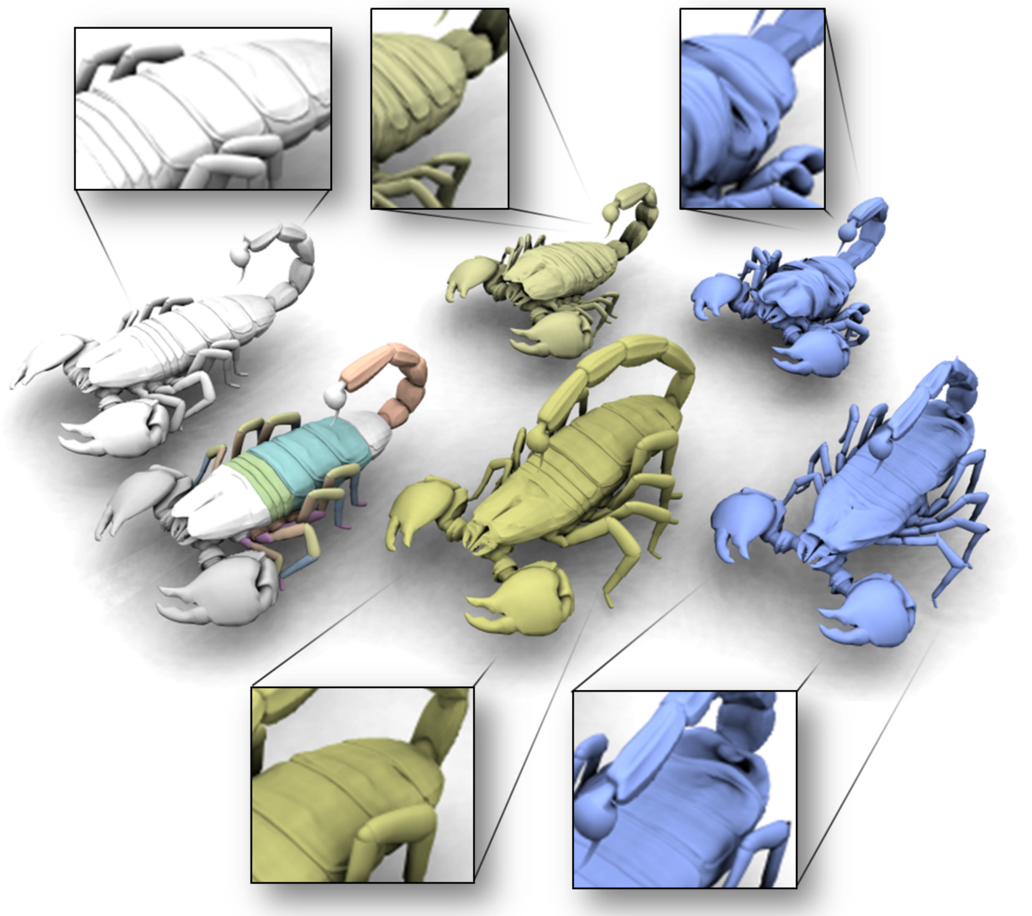


Figure 5.2: *Scorpion model MPM (**M**orphable **P**art **M**odel) shape manipulation results: Lower left (colored parts): original input with user defined parts. Upper left (white): symmetrized mean parts. Middle column (yellow): deformed with our method. Right column (blue): standard elastic (Laplacian/ARAP) surface deformation using exactly the same user constraints. Our structure aware technique fulfills the user constraints without visible distortion, staying in the shape space of parts.*

formulation. We test our framework on a variety of examples enabling interactive, intuitive, and expressive shape manipulations.

5.2 Related Work

Over the last decades, a vast range of research efforts have focused on facilitating shape manipulations under different deformation models, see (Botsch and Sorkine 2008) for a survey. We review only a small selection, focusing on approaches directly related to our goal.

Deformation models: In a highly influential work, Sederberg and Parry (1986) use trivariate Bernstein polynomials for free-form deformation of the embedding volume of an object and thus warp the immersed object. Subsequently improvements have been proposed using richer deformation bases to obtain plausible interpolation behavior of the embedding space ((Lipman et al. 2008; Ben-Chen et al. 2009) and references therein). In an alternate approach, researchers model surface or enclosed volume deformations with approximate elastic behavior using variational formulations, e.g., (Igarashi et al. 2005; Botsch et al. 2006). For surface deformation, popular approaches locally preserve surface details using a Laplacian formulation (Sorkine et al. 2004), or allow deformations that keep local elements as-rigid-as-possible (Sorkine and Alexa 2007). Our algorithm can use any such methods for local deformation. In contrast to other alternatives, we only allow deformations restricted to the learned space of morphable models and conform to the inter-part dependencies using a global coupling.

Statistical shape models: Morphable models, where deformation models are constructed using statistical characterization of the input set of model poses, have been used extensively in computer graphics and computer vision ((Cootes et al. 2001; Hasler et al. 2009) and references therein). Based on available correspondence across the various input poses, the methods perform global dimensionality reduction to compactly encode dominant shape variations. For example, mesh-based inverse kinematics by Sumner et al. (2005) uses a global PCA model to guide deformation modeling. Our method generalizes this ideas to multiple coupled pieces. Modal analysis has also been used to speed up the simulation of deformable objects (Barbic et al. 2009). Feng et al. (2008) use kernel canonical correlation analysis to control animations with a small number of handles, while Baran et al. (2009) present an animation modeling technique by building local deformation models for patches to transfer animations across classes. All of these methods assume global correspondence information across inputs. Zhang et al. (2004) propose FaceIK to combine example faces by spatial weighting and similarity-based weighting and avoid addressing how to combine local PCA models, which is the focus of our work. Tena et al. (2011) introduce a data-driven approach to learn a piecewise PCA face model from facial motion capture data to enable local control for facial expression generation. Our model differs as follows: (i) We model rules for connecting parts such that we can describe a large class of shapes with different arrangements of parts; (ii) Our model is based on an elastic deformation model that puts parts together seamlessly, without need for interpolation and with explicitly modeled interaction between cliques of parts.

Structure-aware deformation: A number of *structure-aware* deformation models exist that try to understand the structure of the input geometry in order to deform the input in a smart way, e.g., Kraevoy et al. (2008) look at differential surface properties to protect vulnerable parts against unnatural bending, while Xu et al. (2009) infer joint properties using a slippage analysis. The iWires system (Gal et al. 2009) uses *wires* or feature curves to learn and maintain intra- and inter-wire relations extracted from man-made objects in order to enable natural deformations. Improvements have been proposed using symmetry hierarchies (Wang et al. 2011) or component based deformations (Zheng et al. 2011). In contrast, we exploit correspondence between potentially strongly deformed parts, rather than parts related only by rigid transforms.

5.3 Part Based Modeling

In this section, we describe our part-based morphable shape model. We first show individual parts model building and then connect these through docking sites. Finally we join the all parts by learned continuous docking properties.

We assume that the input is segmented into parts by the user. This is an easy task done by painting semantic parts in the same color or drawing part boundaries (Figure 5.2, 5.3 or Figure 5.18). Each part is associated with a *part type*, denoted by $\mathcal{T}_1, \ldots, \mathcal{T}_n$. We assume that dense correspondences are available across all the parts of the one type $\mathcal{T}_i$ in the form on consistently connected triangle meshes for all instances. For now, we assume that these correspondences are known, see Section 5.5 for a detailed discussion of the MPM preparation steps.

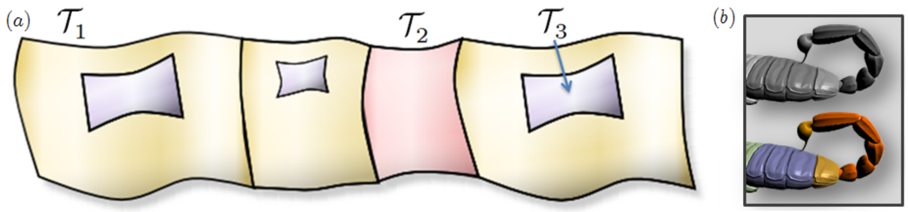


Figure 5.3: *User input for creating an MPM: The input shape is segmented in parts. Same semantic parts are annotated with the same color. (a) abstract view, (b) mesh example. We assume that dense correspondences are available between the parts of one type $\mathcal{T}_i$.*

5.3.1 Single Parts

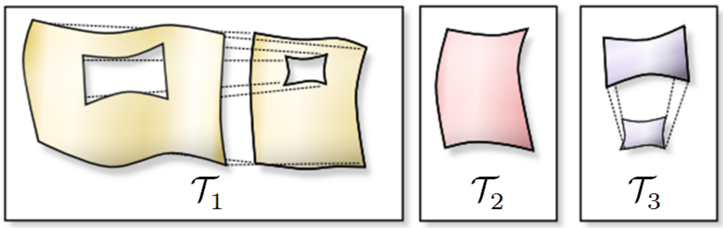


Figure 5.4: *We compute a morphable model for each part type $\mathcal{T}_i$.*

Now we consider a morphable model for one fixed part type $\mathcal{T}_i$ and discuss how it is computed. Since each part of one type has a fixed mesh connectivity with n_v vertices, we encode the different instances simply by varying the vertex positions, denoted as $\mathbf{V} = (\mathbf{v}_1, ..., \mathbf{v}_{n_v}) \in \mathbb{R}^{3n_v}$. Our key observation is such instances are strongly correlated and the $3n_v$-dimensional shape space can be represented compactly by a set of d linear parameters $\Lambda := \{\lambda_1, \ldots, \lambda_d\}$ with $d \ll n$, and a global transformation matrix $\mathbf{T} \in SE(3)$ permitting rotations and translations:

$$\mathbf{V} = \mathbf{T}\left(\mathcal{P}_{\boldsymbol{\mu}} + \sum_{k=1}^{d} \lambda_k \mathbf{b}_k\right) \tag{5.1}$$

where, $\mathcal{P}_{\boldsymbol{\mu}}$ is the mean shape and $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ is a set of orthogonal vectors in $\mathbb{R}^{3n_v}$ that spans the affine subspace of part type $\mathcal{T}_i$. Note that each part type has a separate subspace (Figure 5.4).

Building the part model: Given meshes in correspondence, we construct such a representation from example shapes $\mathbf{V}_1, \mathbf{V}_2, \ldots$ as follows: Using the known correspondences, we first rigidly align all the meshes to the first shape $\mathbf{V}_1$ and apply principal component analysis (PCA) (see Section 4.3 for details). We keep only the dominant eigenmodes; in our implementation we cut off modes with an eigenvalue smaller than 10% of the largest. The eigenvalues of the PCA also provides standard deviations $\sigma_1, ..., \sigma_d$, which act as soft bounds for the parameters Λ.

Variational formulation: Equation 5.1 captures the *ideal* geometry according to the learned model. It is, however, not usually possible to draw shapes exactly from this space, as there may be conflicts either with explicit user constraints or with the implicit constraints arising from assembling multiple parts. We therefore set up a least-squares energy that captures the deviation of a model $\mathbf{V}$ from the learned subspace:

$$E_{sub}(\mathbf{V}, \Lambda, \mathbf{T}) := \frac{\sigma_{sub}^{-2}}{n^2}\left(\mathbf{V} - \mathbf{T}\mathcal{P}_{\boldsymbol{\mu}} - \sum_{k=1}^{d} \lambda_k \mathbf{T}\mathbf{b}_k\right)^2 \tag{5.2}$$

which is minimized by varying free variables $\Lambda \in \mathbb{R}^d$ and transformations $\mathbf{T} \in SE(3)$. We constrain the range in which the subspace coordinates Λ lie by adding the energy

$$E_{var}(\Lambda) := \sum_{k=1}^{d} (\lambda_k / 2\sigma_k)^2 \,. \tag{5.3}$$

E_{var} captures shape variations using a Gaussian model with mean $\mathcal{P}_{\boldsymbol{\mu}}$ and standard deviations $\{\sigma_1, \ldots, \sigma_d\}$ along the principal axes extracted from example data. Further, in the orthogonal complement of the subspace, E_{sub} acts as a uniform Gaussian shape model with standard deviation σ_{sub}.

Please note that, without further constraints, equation 5.3 attracts the mean shape of a part as minimum. For our applications described later, we shift this minimum according to the given task (see Section 5.6).

Gradient domain formulation: When user constraints force vertices to leave the learned shape space while neighboring vertices are not influenced using our formulation (Equation 5.2) results in discontinuities and spike artifacts (Figure 5.5).

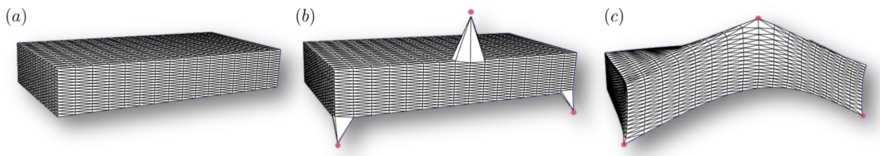


Figure 5.5: *Gradient domain example: (a) We apply three user constraints on a box input model, which can grow or shrink in its longest side length direction. Penalizing deviations of the absolute vertex positions (b), we get discontinuities and spike artifacts. Using the edge vectors in (c), we enable elastic deformations, when leaving the shape space.*

We therefore reformulate the problem as an elastic matching problem in the gradient domain. Instead of penalizing deviations of the absolute vertex positions, we prescribe edge vectors of the triangle mesh. If E denotes the set of all triangle edges, we define:

$$E_{grad}(\mathbf{V}, \Lambda, \mathbf{R}) :=$$
$$\frac{\sigma_{sub}^{-2}}{n^2} \sum_{(i,j)\in E} \omega_{i,j} \left((\mathbf{v}_i - \mathbf{v}_j) - \mathbf{R}\left(\mathcal{P}_{\boldsymbol{\mu}_i} - \mathcal{P}_{\boldsymbol{\mu}_j}\right) - \sum_{k=1}^{d} \lambda_k \mathbf{R}\left(\mathbf{b}_{ki} - \mathbf{b}_{kj}\right) \right)^2 \quad (5.4)$$

where, we weigh each difference vector by a cotangent weight $\omega_{i,j}$ with the angles measured in the mean shape configuration $\mathcal{P}_{\boldsymbol{\mu}}$. We use E_{grad} instead of the direct subspace energy E_{sub}. By measuring the errors locally instead of globally, low-frequency deformations are penalized less, enabling elastic deformations. The free "corotation" variable $\mathbf{R} \in SO(3)$ permits rotations of the parts, which cannot be represented well in a linear subspace model (translations are now handled implicitly by taking differences). Our corotation formulation is similar to (Sorkine and Alexa 2007) with rotations per part rather than per vertex.

5.3.2 Docking Sites

We now describe how to create composite models by stitching together several individual parts. Such an assembly involves a continuous and a discrete aspect. The continuous aspect is what shape the parts should assume such that the geometry fits together, while the discrete aspect decides which parts can be assembled at all.

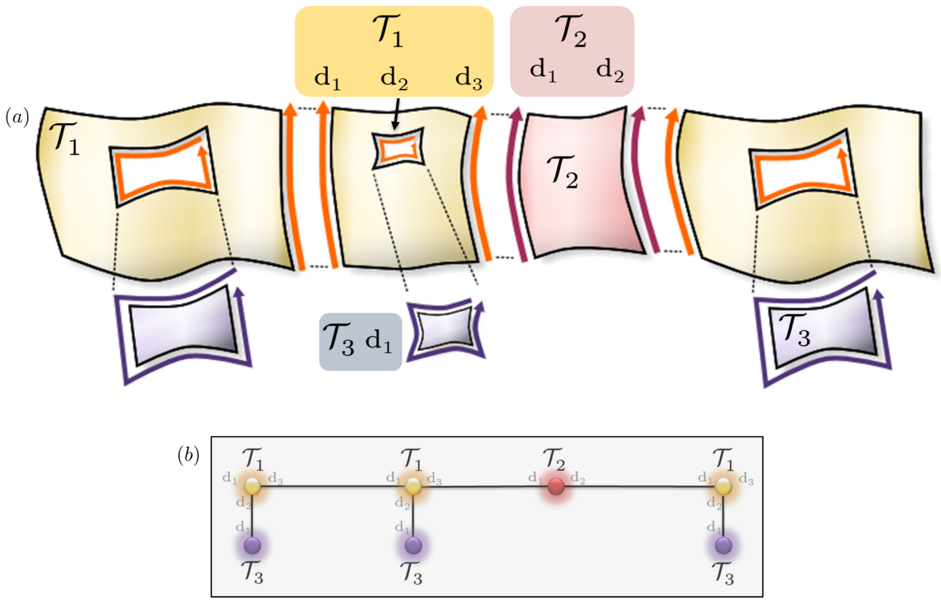


Figure 5.6: *(a) Examining the input model, we tag all connection sites of the morphable parts, where other parts can connect. We also record the types of the possible parts, this leads to a discrete building grammar: For example part type $\mathcal{T}_1$, docker d_2 can connect to $\mathcal{T}_3$ docker d_1. In contrast to that connecting part type $\mathcal{T}_1$, docker d_2 to $\mathcal{T}_2$ is not possible.*
Please note, that our dockers are non-rigid and we have also to model their behavior as described later. (b) Abstract view of the derived part connection graph.

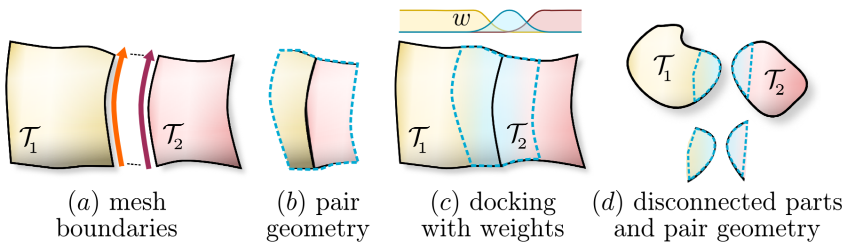


Figure 5.7: *Pairwise docking: (a) A docking site joins parts continuously along by sharing variables along the boundary. (b) We extract geometry adjacent to the boundary and (c) blend between part and pair geometry models for smoother transitions. To be also able to handle non-watertight input models, we allow docking between unconnected part types (d).*

Continuous assembly model: The continuous model describes the interaction of pairs of connected parts: Let us consider a pair of parts $\mathbf{V}_1, \mathbf{V}_2$ of different types $\mathcal{T}_1, \mathcal{T}_2$ that is connected by a common boundary curve (see Figure 5.7a). Clearly, the minimum requirement for joining the two parts is continuity: the two parts should meet seamlessly at the boundary. We enforce this condition by sharing variables among $\mathbf{V}_1, \mathbf{V}_2$, i.e., for coinciding points, we only use single variables when optimizing Equation 5.4. To create the shared variables, we subdivide the corresponding boundary curves and use linear interpolation of the vertices of the eigenvectors of the shape space when subdividing edges. As a generalization of (Bokeloh et al. 2010), we call such morphable boundary curves with shared variables *docking sites*.

Although the shape of the boundary curve transports information between pairs of parts, it only captures limited information on the correlation between the part shapes. We therefore learn a more expressive model from the input data: We form an extended region by gathering the geometry within a fixed distance to the boundary between the pair of parts, called *pair geometry* (Figure 5.7b). As the parts are in dense correspondence, we have dense correspondences between all pair geometry that connects the same part type through the same docking site. We again build a morphable part model for the pair geometry according to Equations 5.3 and 5.4. As before, the parameters are learned from the input examples via PCA. We add the additional energy to the overall energy for all docked part pairs.

To avoid discontinuities, we use smooth weights for all singleton part and pairwise constraints (Figure 5.7c): The attraction to the shape spaces of the parts fades continuously to zero when approaching the boundaries. Contrarily, the attraction to the pair geometry model grows when moving towards the boundary of the parts. We weight each vertex by $\exp(-d^2/\sigma_{bdr}^2)$, where d is the distance to the boundary.

In real world input meshes, we also encounter models constructed out of separate parts (e.g. Figure 5.15a) not sharing a common boundary and with empty space inbetween them. Therefore, we extend our docking site definition to also allow pairs of parts that are disconnected. We do so by forming pair geometry that includes regions of both models (Figure 5.7 d). Algorithmically, we include all geometry in part 1 that is within at most a fixed distance from part 2 and vice versa. To create the connectivity edges for the part geometry, we connect each such pair of points that connects from part 1 to 2 by a virtual, non-manifold edge and setup the Laplacian deformation model of Equation 5.4 to preserve these distance vectors. This way, the variations in relative pose are automatically learned from the different example configurations.

Discrete assembly model: Assembling parts is possible only when we observe similar connections in the input. Otherwise, we do not have correspondences along the boundary curves, which may even have different topology. Further, assembling arbitrary parts is usually semantically meaningless.

We build a shape grammar to capture this constraint: Given a part decomposition, we first compute all boundary curves. We segment the boundary of each input part into segments where different part instances connect. These segments form the docking sites. For each combination of the same part types across the same pair of docking sites (see Figure 5.6), we build one pair geometry model as described above.

5.4 Computing the Geometry

In order to apply our morphable part model, we now need to address two problems: First, given a discrete graph of docked parts, how to compute a geometric embedding that maximizes the likelihood of the overall model. At this point, we also incorporate user constraints, if any, to permit interactive exploration of shape variants. Second, how to construct graphs with consistently docked parts.

Assume we are given an undirected discrete graph G of part types connected through docking sites. We first assemble a compound mesh of all parts, with vertex variables shared across docking boundaries. We then accumulate the corresponding energies: Let $E_i, i = 1 \ldots k$ denote the singleton energy functions of the individual parts and $E_{i,j}, (i,j) \in G$ the pairwise energies contributed by the docking sites pair geometry. Each energy term $E_i, E_{i,j}$ is the sum of terms $E_{var}(\Lambda)$ and $E_{grad}(\mathbf{V}, \Lambda, \mathbf{R})$ according to Equations 5.3 and 5.4. Thus,

$$E(M) = \sum_{i=1}^{k} E_i(\mathbf{V}, \Lambda_i, \mathbf{R}_i) + \sum_{(i,j)\in G} E_{i,j}(\mathbf{V}, \Lambda_{i,j}, \mathbf{R}_{i,j}). \tag{5.5}$$

The unknowns of this system are: (i) the positions of the vertices $\mathbf{V}$ of the joint mesh, (ii) the respective rotation matrices $\mathbf{R}_i, \mathbf{R}_{ij}$, and (iii) the shape space parameters Λ_i and Λ_{ij}.

User constraints. The variational formulation of Equation 5.5 allows us to easily incorporate additional user constraints. We implement a simple handle-interface that allows the user to click on model points and prescribe their position by adding one or more according quadratic potentials $E_{user} := (\mathbf{v}_i - \mathbf{y})^2$, where $\mathbf{v}_i$ is a model vertex and $\mathbf{y}$ is the desired target position. Thus, the user can implicitly navigate in the joint shape space by prescribing sparse shape constraints resulting in an intuitive and easy to use tool for refining the geometry.

In addition to spatial constraints, we also support placing soft constraints on shape parameters λ_i. In shape modeling, often coupling across non-local parts is desirable. Hence, we allow the user to tag selected parts of the same type to be coupled. We realize this by a quadratic penalty $E_{sym} = (\Lambda_i - \Lambda_j)^2$ for any part pair $\mathcal{T}_i, \mathcal{T}_j$ indicated by the user.

5.4.1 Solving the System

In order to minimize the above energy, we alternate between solving for the vertex positions and shape parameters (i,iii) while keeping the transformations (ii) fixed, and vice versa. Optimizing the vertex positions is done by equating the gradient of the quadratic energy with zero and solving a linear system. The transformation variables are computed by shape matching.

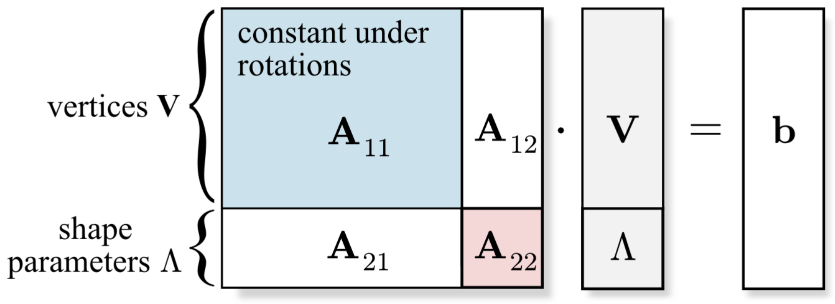


Figure 5.8: *Matrix decomposition. The system matrix consists of a sparse Laplacian matrix* $\mathbf{A}_{11}$ *that remains constant. The matrices* $\mathbf{A}_{12} = \mathbf{A}_{21}^T$ *depend on the rotation variables. We employ a Schur decomposition: We solve the large system* $\mathbf{A}_{11}$ *rapidly using prefactorization. It remains to solve a small linear system of size* $\mathbf{A}_{22}$ *and to perform matrix-vector products with changing matrix* $\mathbf{A}_{12}$*.*

Numerical speedup: The alternating optimization for the Laplacian deformation model proposed in (Sorkine and Alexa 2007) is efficient because the linear system can be prefactorized once; each iteration only requires a very efficient back-substitution step. We cannot prefactor the above linear system, which makes interactivity difficult. Specifically, the basis vectors of the affine shape space are rotated in each iteration, thereby altering the system matrix A. The shape parameters, however, are low-dimensional in comparison to the vertex positions. The matrix of the linear system can be understood as an inverse covariance matrix of the Gaussian distribution in vertex and shape parameter space (see Figure 5.8): We can order the columns and rows such that the (large) upper left block $\mathbf{A}_{11}$ forms the quadric coupling vertices to vertices, while the lower right block $\mathbf{A}_{22}$ couples shape parameters, and the off-diagonal blocks $\mathbf{A}_{12} = \mathbf{A}_{21}^T$ encode dependencies between shape parameters and vertex positions. The matrix $\mathbf{A}_{11}$ is constant under changing transformations

$\mathbf{T}_i$, depending only on the given constant mesh connectivity. But $\mathbf{A}_{12} = \mathbf{A}_{21}^T$ has to be recomputed, because the shape basis vectors $\mathbf{b}_k$ are co-rotated by the $\mathbf{R}_i$ in Equation 5.4, thereby changing matrix entries associated with the shape parameters. We therefore split the solution using a Schur complement decomposition (see (Boyd and Vandenberghe 2004), pp. 672ff).

In this decomposition we solve the large, constant system (matrix $\mathbf{A}_{11}$) efficiently using a precomputed sparse Cholesky decomposition and solve the small problem using conjugated gradients. Using this approach, we typically observed an order of magnitude speedup in comparison to a solution without prefactorization.

We apply the described solver to compute vertex positions given the linear system derived by an input graph of parts and pair geometry. In the next section, we show ways to establish such a graph of parts.

5.4.2 Creating Part Graphs

In order to manipulate the discrete component of our morphable part model, we need to create part graphs that are consistently docked, i.e., docking sites must match to exactly one counterpart of a matching type. We show two ways to do that: The first one are editing operations on a given graph and the second automatically building a valid tree structured graph from scratch.

In the interactive editing of graphs, the user can delete from or insert parts into a given graph. We start for example with one graph of the input models. Figure 5.9 and Figure 5.16 show results of this approach.

To delete a node, i.e. a part, it just has to be selected. Using the learned grammar, our method automatically checks if the resulting graph will still be valid. Only then the operation is allowed. The new edge between the neighbor nodes is equipped with the appropriate pair geometry connection. Finally, we solve the system, to present the edited shape.

To insert a node, the user selects neighboring nodes. Our system checks the grammar rules and offers all possible parts for this combination. The user selects the desired part type. We automatically select the appropriate learned pair geometry model and insert the node.

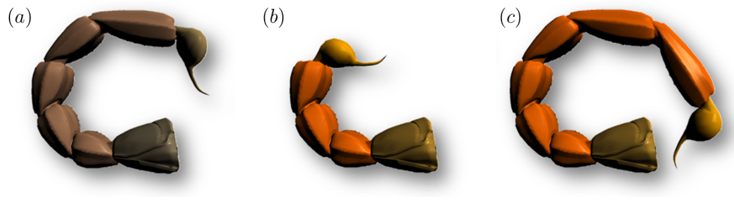


Figure 5.9: *Here we edit a given discrete graph (a): The user deleted two elements (b) or inserted one element (c). The user also assigned the desired continuous parameters for the new Λ_i of the inserted part. Our system automatically computes a geometric embedding for the new graph.*

For the automatic generation of graphs, we assume that all input models are tree structured. This assumption is true for many organic input models like all kinds of animals. Our method is capable of automatically creating all possible graphs compatible to the learned grammar. We start with a random node, for all open docking sites, we randomly apply a valid grammar rule connecting this docker to another part. We repeat this on all open docking sites until all dockers are closed. Figure 5.19 shows examples generated by this approach. For this example the continuous parameters Λ_i of the inserted parts have been randomly sampled.

Seamless stitching: For watertight dockers, we need to ensure that two parts $\mathcal{P}_1, \mathcal{P}_2$ that share a common boundary fit together seamlessly. We do so by sharing the vertices along the boundary curve, i.e., using only a single unknown variable for each shared vertex. As we have dense correspondences along the two boundary curves, we take the union of the vertices and split the triangles accordingly. The statistical model for newly inserted vertices from $\mathcal{P}_2$ is obtained by interpolation of the two closest vertices from $\mathcal{P}_1$, and vice versa.

5.5 Preparations for a MPM

For the utility of our approach, it is essential that also inexperienced users are able to convert arbitrary 3D models or model collections into an MPM.

In Chapter 4, we presented an automatic approach to identify meaningful parts in a given model. This can be seen as a first step for the unsupervised creation of an MPM, but is not yet capable to achieve this challenge unsupervised for all classes of input models. One reason for this is that it is bootstrapped by features not present on all geometry classes. Further, semantic knowledge is needed to select the parts in some cases.

On account for this, we designed our part definition system as an interactive tool. A user just paints parts of the same type in the same color. If two parts of the same type have a common boundary, we allow to separate the parts by selecting a border line. After this process we obtain a user segmented input mesh.

5.5.1 Automatic Generation of Correspondences

Now we have to establish semantic dense correspondences between parts of very different geometry, given rough user segmentations. As we showed in the last chapter, this is not possible for all classes of arbitrary input data using our feature-based symmetry detection scheme.
We consider this task as an independent problem. The focus of this chapter is the morphable part model rather than the computation of semantic correspondences. In (Burghard et al. 2013) we describe an automatic approach for computing dense correspondences of the parts, given the described user input only. The method is not part of this thesis and we only describe it very briefly in this section.

In a first step, the parts are extracted out of the input mesh, by cutting at the color boundaries. They are grouped into part types $\mathcal{T}_i$ identified by their color. For each group we first heuristically parametrize the boundary curves and then the whole part. Using this, we transfer the mesh connectivity of one part to all others of this type. We have now one mesh with different vertex positions for every part instance. These heuristically computed correspondences ensure only to map to the target surface, but not to necessarily establish a reasonable shape space.

Tangential ambiguity along the target surface leading to different drift in each pairwise correspondence distorts the morphable model. When building the PCA model out of these correspondences, it is not unlikely that parametrization differences dominate the geometric variations, thereby polluting the PCA space. In fact, shape spaces for the parts created by correspondences of this simple, heuristic method result in very poor results. Therefore we optimize the correspondences in a next step.

Researchers in computer vision have investigated measures for *good* Gaussian shape spaces (Hill and Taylor 1994; Kotcheff and Taylor 1998; Davies et al. 2002). One way to characterize compactness of a Gaussian model uses its entropy, which is related to the determinant of the covariance matrix (Kotcheff and Taylor 1998). We adopt this approach and this regularizer removes unnecessary variance that is not justified by data matching.

The actual optimization is carried out in the tangent space of the shapes: We perform gradient descent on this entropy based energy function. The correspondence optimization is useful in removing artificial drift in the correspondences and obtaining concise Gaussian shape models. Please see (Burghard et al. 2013) for details.

5.5.2 Learning Continuous Parts

Having dense correspondences for the parts now, learning Gaussian shape models is straightforward: we simply apply principal component analysis (PCA) to the set of input parts of one type in the optimized correspondences and store the eigenmodes and variances for our MPM. Subsequently, we compute the shape parameters for each example part.

5.5.3 Learning Docking Rules

After establishing the continuous model for each part type, we learn a shape grammar, i.e., a set of discrete rules as how the parts can be attached to each other. Since we already have a graph decomposition of the input model (see Figure 5.6), we simply search and collect patterns of how each node in the graph is connected to its 1-ring neighbors and store them as a rule set.

Between two docking parts P_i and P_j, we identify the docking area based on an "influence region" parameter r — we include all points of part P_i that are within distance of any point in P_j, or vice-versa (see Figure 5.7). We then apply PCA on every pair geometry and store modes and variances as for the continuous parts.

5.6 Applications

We employ the proposed part-based morphable model in the following application scenarios: As a tool to enhance *free-form deformation* results on single models, for *partial symmetrization* of semantically corresponding parts, and as a tool to extend *inverse procedural modeling* algorithms e.g. (Bokeloh et al. 2010) to deformable parts with correspondences based on semantics guided user input rather than rigid matching. Extending this from one to many input models (segmented by the user analogue to a single model) represents another scenario: Here we let our method creatively design new shapes from examples.

5.6.1 Free-Form Deformation

In our variational formulation for embedding graphs of parts, it is easy to enable free-form shape deformation using additional energies to model handle constraints.

Note an important detail: If we use the part-based model as is, the original state of the model is not the optimum of the energy function but the model will be biased to deform towards the mean in each part and each docking site. This is not desirable for a free-form deformation tool, where the rest state should always be the original model. We therefore exchange the means: For each actual part instance, we determine the shape parameters λ within the learned subspace and replace the learned average mean with the actual part parameters $\lambda_{k_{observed}}$, thus removing the bias:

$$E_{var}(\Lambda) := \sum_{k=1}^{d} \left(\lambda_k - \lambda_{k_{observed}}/2\sigma_k\right)^2. \tag{5.6}$$

Using our technique, we also provide some more advanced editing features: We can place additional constraints on subspace coordinates: In Figure 5.13,

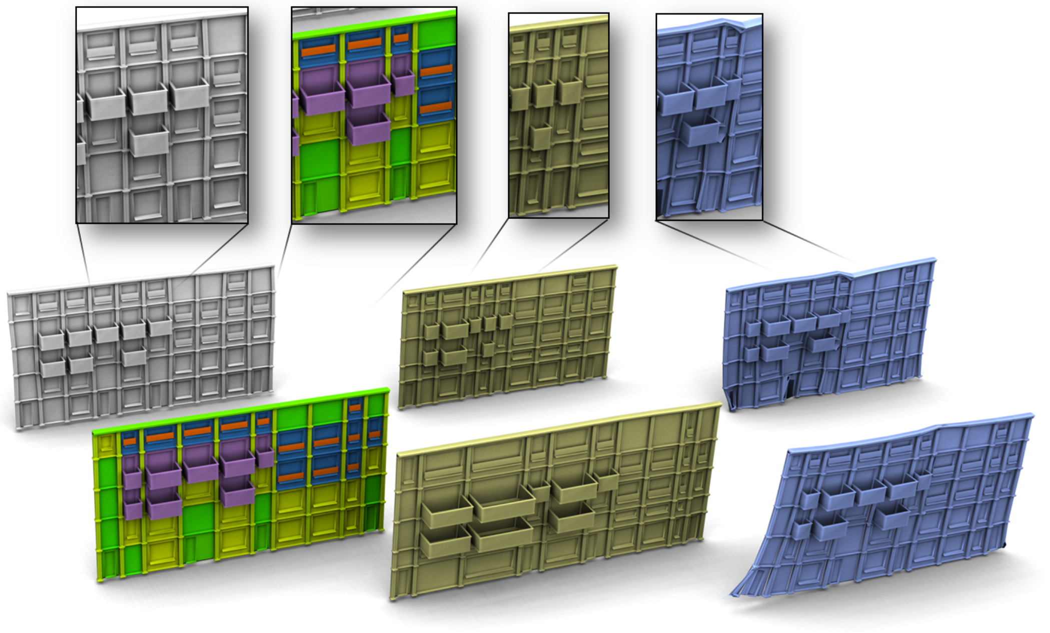


Figure 5.10: *Facade model editing results: Lower left (colored parts): original input. Upper left (white): symmetrized. Middle column (yellow): deformed with our method. Right column (blue): standard elastic (Laplacian/ARAP) surface deformation. Our structure aware technique fulfills user constraints without visible distortion while staying in the shape space of parts.*

we enforce similarity of local subspace coordinates. Nearby leaves' Λ coordinates have to behave similar to the edited leave that is influenced by free form deformation. In this case we simulate a stronger and weaker growth of one leave by dragging it bigger and smaller – nearby parts react similar without touching them.

In another example (Figure 5.14) we couple the coefficients of all yellow parts. The user already fixed a few points at the red dots and moves then one of the handles, which forces a yellow element to modify its shape space coordinates. The coupling influences the coordinates of the other yellow parts that mimic the behavior of the edited part.

Employing this technique is a powerful tool in model editing. Our extensions manipulating Λ coordinates show, that there are more promising editing possibilities enabled by our MPM model that we leave for future work.

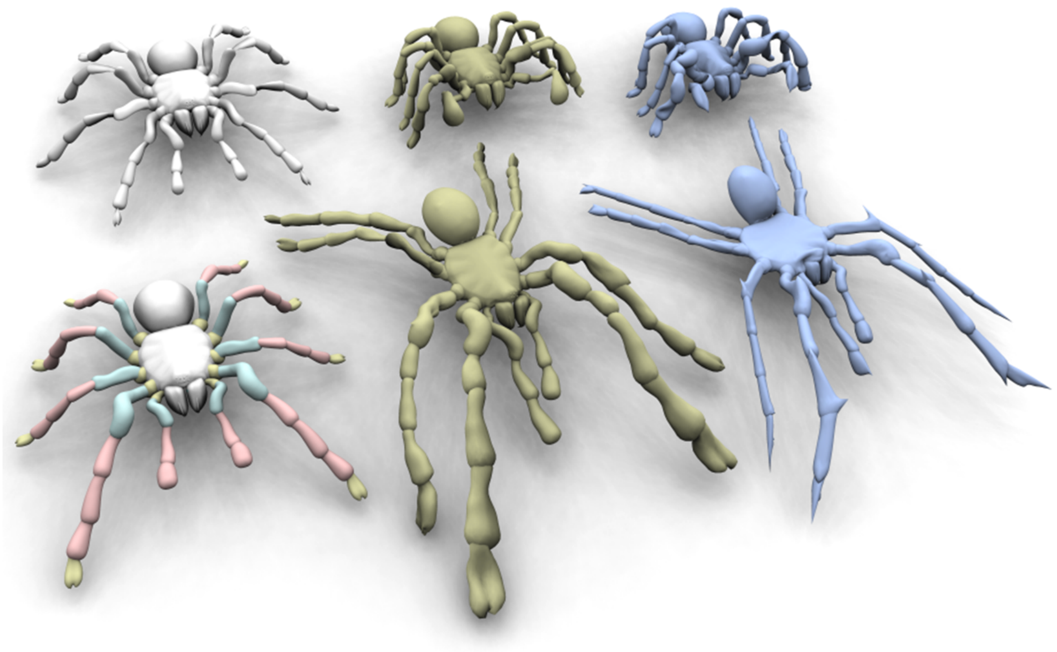


Figure 5.11: *Spider model editing results: Lower left (colored parts): original input. Upper left (white): symmetrized. Middle column (yellow): deformed with our method. Right column (blue): standard elastic (Laplacian/ARAP) surface deformation. Our structure aware technique fulfills user constraints without visible distortion while staying in the shape space of parts.*

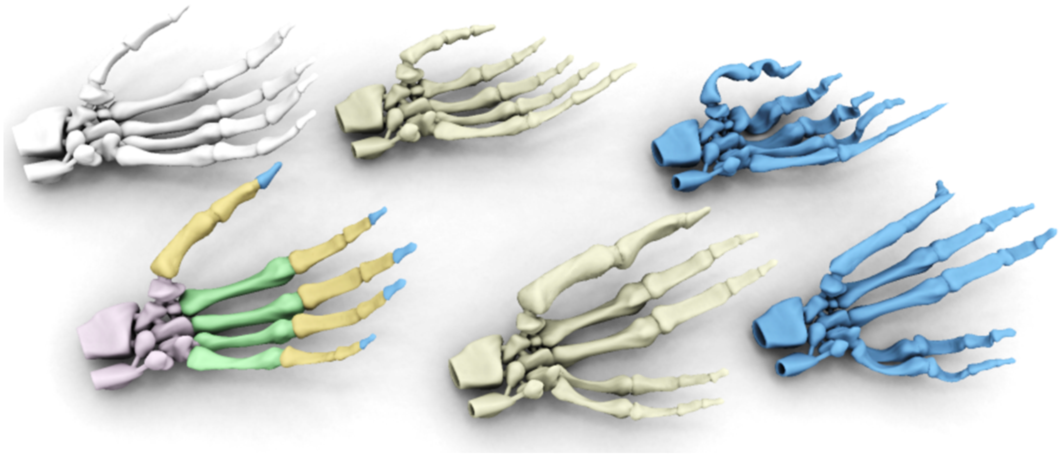


Figure 5.12: *Hand model editing results: Lower left (colored parts): original input geometry (courtesy of Georgia Institute of Technology). Upper left (white): symmetrized. Middle column (yellow): deformed with our method. Right column (blue): standard elastic (Laplacian/ARAP) surface deformation. Our structure aware technique fulfills user constraints without visible distortion while staying in the shape space of parts.*

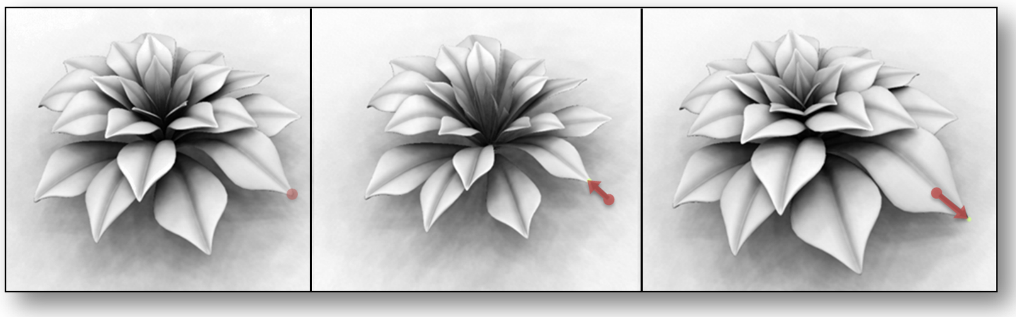


Figure 5.13: *Additional constraints placed on subspace coordinates: Plant model: enforcing similarity of local subspace coordinates.*

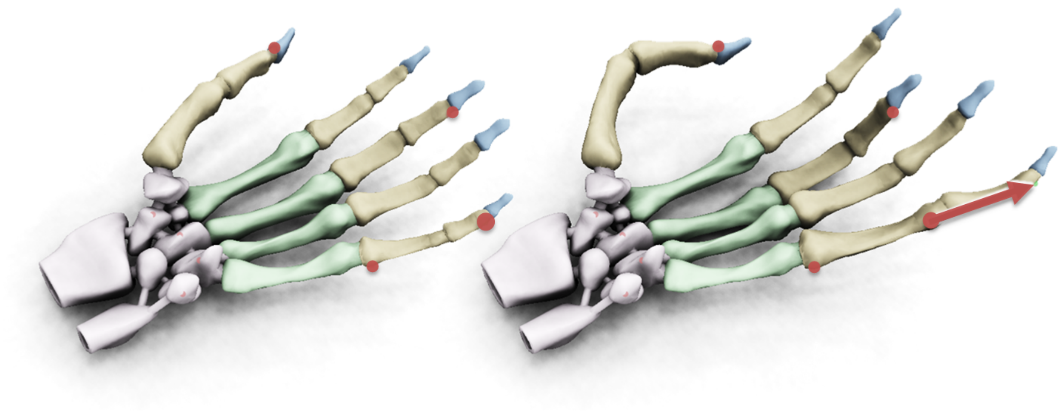


Figure 5.14: *Additional constraints placed on subspace coordinates: Hand model: editing with global influence of the yellow parts.*

5.6.2 Partial Symmetrization

In this application, we set the average mean for all part classes and replace the original covariance matrices with isotropic Gaussian distributions with small variance. This encourages the individual parts to assume the same mean shape in each part. By changing the mean within the available subspace parameters, we can control the shape that all the parts are trying to assume (see Figure 5.15). The opposite effect can also be created, by increasing the distance from the per-part mean, which creates caricatures of the input model (as done by Blanz and Vetter (1999) for single component models).

5.6.3 Inverse Procedural Modeling

By training a part-based morphable model, we construct not only a deformation model but also a set of discrete rules for assembling the parts. The continuous deformation model serves as a tool to compute an actual embedding of the abstract graph, while the residual energy of the optimal embedding indicates the overall distortion necessary to realize the graph.

As described, our approach automatically learns a grammar according to which parts can be attached. We allow the user to manually specify a new graph based on the constructed grammar. Subsequently, we run the continuous optimization to compute its optimal embedding.

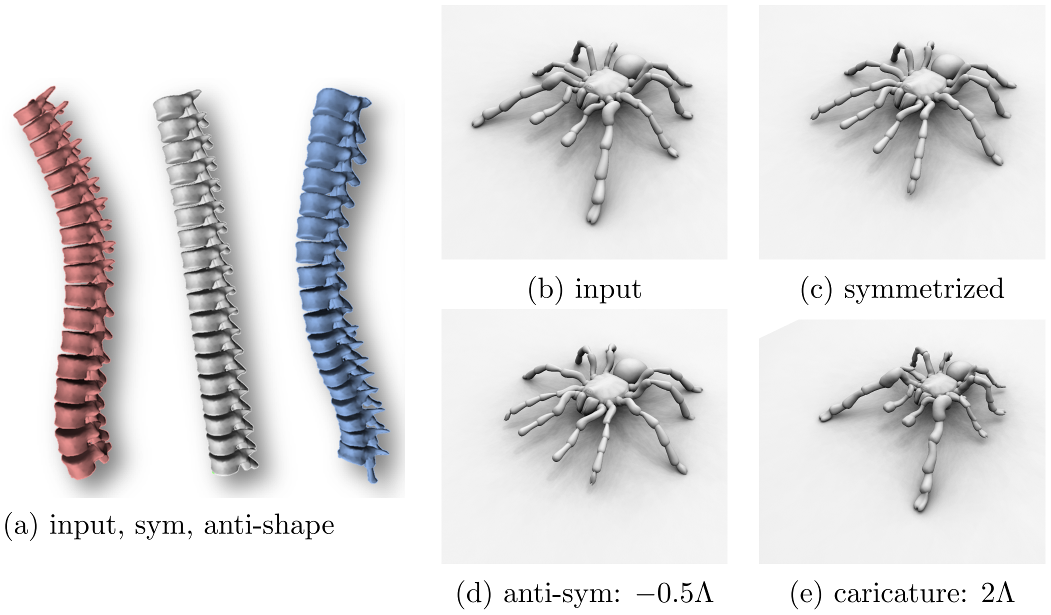

(a) input, sym, anti-shape
(b) input
(c) symmetrized
(d) anti-sym: -0.5Λ
(e) caricature: 2Λ

Figure 5.15: *Symmetrization by setting mean subspace parameters. Caricatures are obtained by doubling the subspace coordinates. Using negative values yields an "anti-shape".*

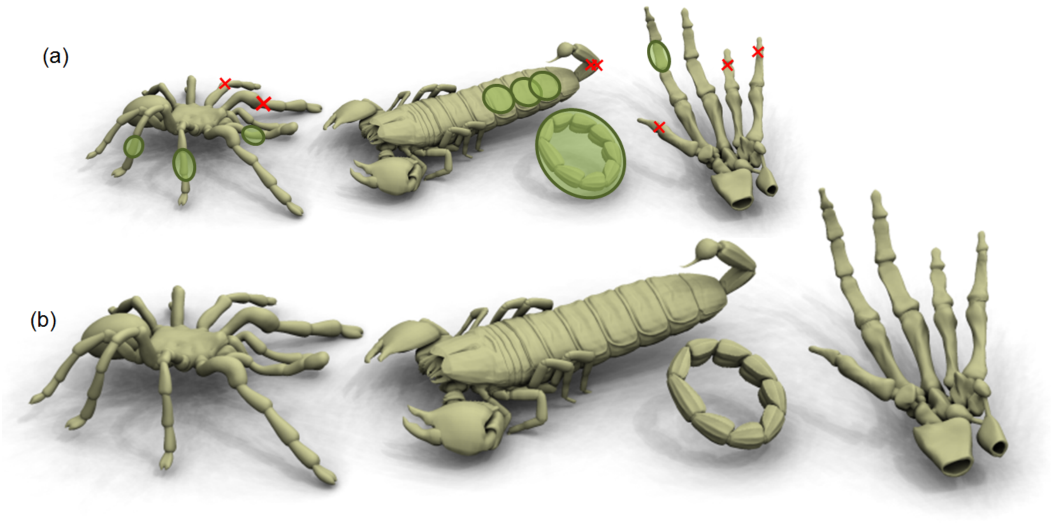


Figure 5.16: *Changing the discrete graph structure, assembling parts in a different configuration and solving for optimal embedding. (a) Visualization of inserted (marked green) and deleted (marked red) elements. Spider: adding/removing segments to the spider legs; scorpion: extending the scorpions body and removing parts from the tail; below this: creating a cyclically connected tail; hand: inserting a segment to the pointing finger and removing parts of the remaining fingers. (b) Results without marks.*

model	**vertices**	**parts**	**def. time(ms)**	**type**
hand	23081	20	150	bnd
facade	45493	128	300	cmp
tarantula	28996	57	170	cmp
scorpion	66453	59	350	bnd & cmp
spine	16254	18	150	cmp
hexa-grid	51465	15	350	bnd
plant	22330	29	180	cmp

Table 5.1: *Model statistics. Timings: average for one iteration. Type: boundary docking sites (bnd), disconnected components with generalized docking sites (cmp), or both.*

5.7 Implementation and Results

We have implemented our method single-threaded in C++, but using the multi-threaded Intel Math Kernel Library for solving linear systems. Table 5.1 lists performance statistics on a workstation with an Intel quad core i7-2600K (3.4 GHz) and 8GB of RAM. Results are shown in Figures 5.1, 5.14, 5.15, 5.16.

We used the following models: The *hand* model is a manifold meshes with boundary docking sites, while the *plant*, the *tarantula* and the *spine* models consist of separate parts that are connected by generalized docking sites. The *scorpion* model uses both techniques - boundary docking sites in the body region and separate parts in legs and tail. In addition, we synthesized the *facade* and *hexa-grid* to complement the evaluation set. For the models from external data sources, we estimate correspondence as described in Section 5.5.1, while for the synthesized models we use known correspondence information.

Figures 5.10 – 5.14 show deformation results obtained with our technique. The model in the lower left corner is the original input, with corresponding parts tinted in matching colors. The brown variations are editing results from sparse constraints and the white model shows a symmetrized variant.

We use the as-rigid-as-possible Laplacian surface editing model (Sorkine and Alexa 2007) as representative for elastic free-form deformation techniques. This can also be seen as our method, without the structure information known from dense correspondences between parts. The according results are shown in blue. In comparison to MPM results, we observe artifacts: (i) When models are squeezed, unlike our data-driven approach, elastic deformation produces unfavorable results with surfaces folding and wavy artifacts (e.g., hand and tarantula examples). (ii) When models are stretched, the results, in absence of folding, are still not plausible in comparison to the learning-based results.

Note that unlike Sumner et al. (Sumner et al. 2005), we are able to learn the shape space for these results from a *single* model by exploiting part-level symmetries using inter- and intra-part correlations.

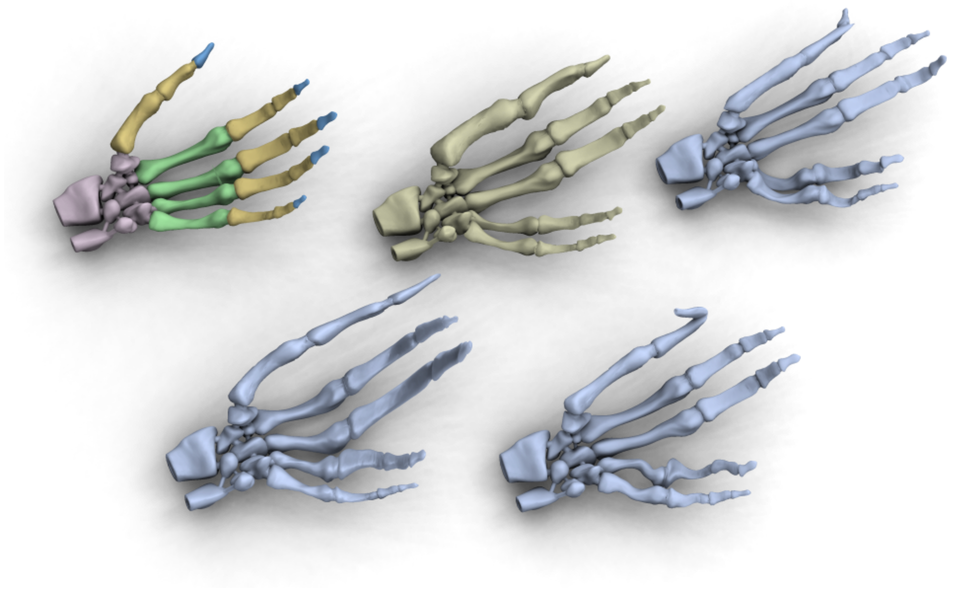


Figure 5.17: *Comparison to standard deformation techniques - clockwise from upper left: original input, our result, Laplacian surface editing, finite-element based elasticity, thin-plate-splines.*

We also compared MPM to a larger set of previously proposed structure unaware deformation techniques (see Figure 5.17): We tested a finite-element-based elasticity model (Adams et al. 2008), which behaves similar to the Laplacian surface editing, including all the artifacts. A thin-plate spline deformation model (Brown and Rusinkiewicz 2007) that fits a smooth deformation field rather than minimizing stretch avoids the stretch and wrinkling artifacts, but the results still look unrealistic. In particular, thin-plate-splines permit arbitrary affine mappings at no cost which leads to distorted results (anisotropic squeezing and expansion).

We also tested discrete modifications to the example models, setting a custom part graph consistent with the established shape grammar (see Figure 5.16). Our solver assembles the pieces together in plausible configurations as if the shapes had been designed like that, even in presence of cyclic dependencies.

Figure 5.13 and Figure 5.14 show the effect of directly prescribing subspace coordinates for the parts: In Figure 5.13, subspace coordinates are diffused to their neighbors during editing to encourage a locally symmetric behavior, while Figures 5.14 shows global coupling of coefficients over the model. Distributing lambda constraints is used interactively, with the force derived from user constraints.

5.8 Application on Shape Collections

To represent the morphable part model space of shapes spanned by a example set of complex geometry, we apply our model on multiple related input meshes. We use exactly the same techniques described up to now, just replacing our single input example with an arbitrary number of consistently marked input examples. We build the statistical model of the individual part types and their pairwise connections across the models and learn the discrete combinatorial model out of all graphs of parts.

We create shape variations of the input models by constructing combinatorially valid part graphs and subsequently computing smooth geometry that maximizes the likelihood of parts and their connection. We apply our model to create shape variations by either interactive editing with sparse user constraints, or fully automatic sampling from the shape space.

The random sampling of shapes is a useful tool for automating model creation, for examples, for large amounts of background props or creative new shapes and for exploring the shape space. To obtain a random sample shape, we first create a discrete part graph by randomly choosing grammar rules learned from observed graphs, with likelihood according the frequency of occurrence in the training data. Next, we sample for every shape separately the continuous Λ parameters according to their learned σ and compute a geometric

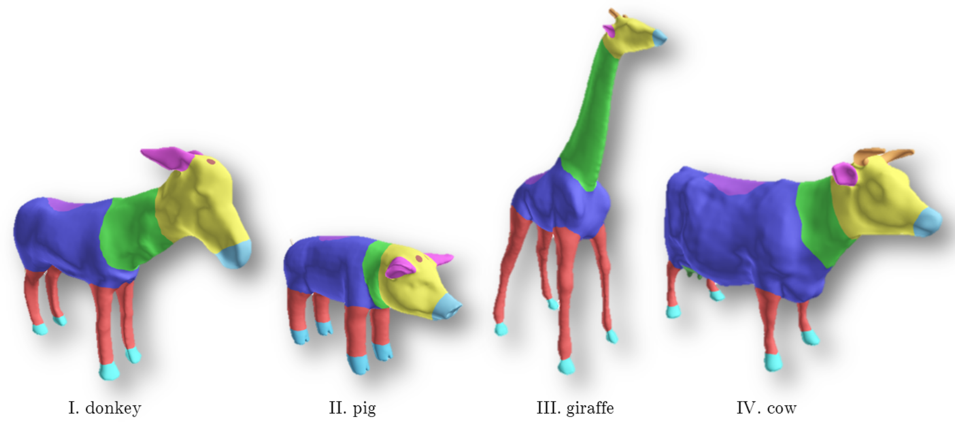


Figure 5.18: *Application on shape collections: We demonstrate our method on multiple input models: These four mammals are segmented by the user to semantic parts, just by painting with the same color. We compute dense correspondences between the parts. Instead of using one input model only, now this whole scene represents the input for our MPM system. Data sets provided by (Giorgi et al. 2007).*

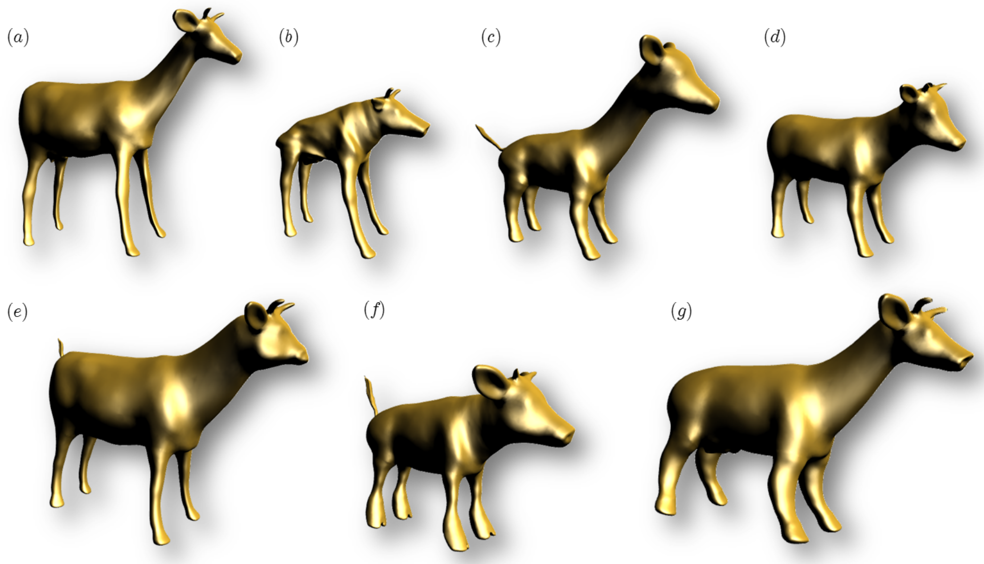


Figure 5.19: *These models were automatically sampled from the MPM-shapes of given in our input. Please note, that result (a) looks like a deer while (d) is similar to a calf model. Both were not given in the input data. The user can now interactively tune the models using our system. All created shapes are of high mesh quality, watertight and can directly be used e.g. in games or for 3D printing.*

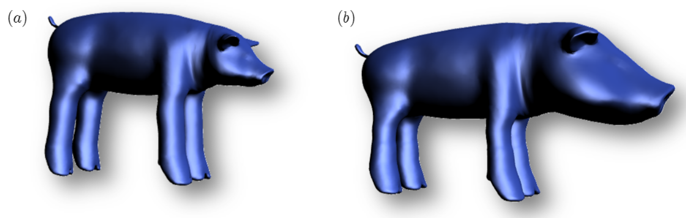


Figure 5.20: *Results of our system with scaling parts only: Here we use our system without correspondence information between the parts. Parts and pair geometry can only scale uniformly as we cannot compute morhpable models. We still achieve smooth, watertight results, but our impression is that they are less creative than combined results. Further, all the parts must be taken from one input example as no universal pair geometry can be learned without correspondences.*

embedding.

To account for relative scaling in the input models, we add an artificial uniform scaling mode to the shape space of each part. For the random sampling of parameters, the variance of this artificial mode is set by the user.

We apply our method to a shape collection of mammals (Figure 5.18) and demonstrate that this approach yields more meaningful shape variations than a part-based method that did not learn shape statistics (Figure 5.20).

Our result models in Figure 5.19 were automatically sampled from the MPM-shapes resulting from our input shapes. Please note, that certain randomly sampled results appear like real existing animals (deer and calf result), although both were not given in the input data. The user can now interactively fine tune the models using our system or change them completely by resampling parameters or employing discrete edits. The created shapes are of high mesh quality, watertight and can directly be used e.g. in games or for 3D printing.

5.9 Limitations

In our framework, we require dense correspondences for the parts. If the automatic method (Burghard et al. 2013) we use fails, it is difficult to establish these correspondences, requiring much user interaction manually setting landmarks for correspondences.

Second, our framework only applies to objects with a clear part structure with one-to-one correspondence: we cannot capture certain types of redundancy such as fractal terrains or irregular textures such as bark of a tree.

Finally, in highly complex models, say a detailed car model consists of many tiny parts, a hierarchical decomposition is desirable. Potentially a coarse structure (or cage) can be used in this regard.

5.10 Summary

We presented a novel technique for shape deformations based on morphable parts. Instead of using predefined differential shape priors such as elastic (as-rigid-as-possible) or continuous (thin-plate splines) behavior, we learn the variations from correspondences across symmetric parts from a single input example or shape collections. Our model represents shape variations of parts, and pairwise relations of connected parts.

With only a sparse set of user constraints, we enable plausible deformations. In contrast to previous techniques based on morphable models, our approach utilizes partial symmetry to extract more information from the input data and learns rich shape variations from small amounts of training data.

The real-time performance allows an interactive exploration of shape variants using handle-based constraints. Discrete variations are created by a graph mutation method.

We explored applications such as partial symmetrization, caricatures, and presented first steps towards generalized inverse procedural modeling by learning a deformable shape grammar and reassembling parts accordingly. We showed a system that allows to explore numerous models inspired by a few example shapes. Exploring model variation requires only minimal user input and yields plausible results. We believe that techniques for extracting and representing the knowledge contained in model collections is a very important problem, with potential to substantially influence the field by facilitating reusing of 3D content.

6 Conclusion

In this work, we have shown algorithms that discover shape varying repetitive structures in given unstructured data and presented ways to exploit this information.

Discovering redundancies: 3D scans of large (e.g. house facades) or small (e.g. statues) objects frequently provide a rich pool of redundancies in the geometry. But considering a raw scan, all we have is only a long list of vertex positions. Although a human is able to identify even complex redundancies by just looking at the rendered scan with one glance, it is a hard problem computing equivalent results fully automatically, even worse if the repeating shape varies intensively.

Our approach to this problem is – and we assume humans think analogously – to first identify salient features on the geometry and then examine the constellation of them. In this concept we introduced our method working on rigid mappings between the repeating parts and later extended it to more general mappings.

To find a similar constellation of features, we introduced a subgraph matching algorithm. We first showed that such an approach can actually perform reliable rigid symmetry detection. In contrast to previous work, our new method works without restrictions to regular patterns or nested hierarchies.

Second, we presented the generalization to isometric matching using geodesic lengths and angles on surfaces instead of rigid Euclidean transformations as validation criterion. Our method can handle partial isometric symmetries, as well as more general symmetries where the preservation of intrinsic geometric quantities can be relaxed.

The most important disadvantage of this method are the parameters for the matching tolerances that are difficult to adjust. The user still has to define how intense a non-isometric deformation may be in order to be recognized. For unknown data, too small tolerances result in no matches at all, while too large tolerances produce many false positives. This issue has its origin in the question what is still symmetric and what is too deformed to be recognized as the same part.

Our idea here is to learn the allowable mapping functions from the data. Doing this in an unsupervised way for arbitrary input data is a very challenging task. We introduced subspace symmetries to capture similarity between surface geometry, which are related by non-rigid transformations that are not arbitrary but span a low-rank subspace. The resultant symmetry subspace then has a natural compact description, and effectively captures the variations of the underlying surface. We found many man-made models to be a rich source of such symmetries, as shown in our experiments.

Employing redundancies: Exploiting known symmetry correspondences, we showed a wide range of applications to use this information, namely amongst others simultaneously editing of geometry, symmetry-based reconstruction of rigid parts and compression of large scanned scenes.

We extended these techniques for non-rigid symmetries. Working on the extracted symmetry subspaces we showed non-local non-rigid denoising, model completion and simultaneous instance replacement, while factoring out the underlying subspace variations.

Finally we presented a novel technique for shape deformations based on subspaces. Instead of using predefined differential shape priors such as elastic or continuous behavior, we learn the variations from correspondences across repeating parts from a single input example or from shape collections. Our model represents shape variations of parts, and pairwise relations of connected parts.

With only a sparse set of user constraints, we enable plausible deformations. In contrast to previous techniques based on morphable models, our approach utilizes partial symmetry to extract more information from the input data and learns rich spaces of shape variations from small amounts of training data.

We explored applications such as partial symmetrization, caricatures, and presented first steps towards generalized inverse procedural modeling by learning a deformable shape grammar and reassembling parts accordingly. We showed a system that allows to explore a large space of models spanned by a few example shapes.

6.1 Future Work

Our method depends on the concept that structural redundancies result in recognizable constellations of features. In our experiments we found that not all classes of input data can be handled with one feature type (e.g. points or crest lines). So, for future work it would be interesting to employ alternate feature types. Further it may be possible not to use special feature points or regions, but just sample the surface uniformly and work on the descriptors at the samples. It will be interesting how that can robustly initialize and capture subspace symmetries.

For the deformation framework, it would be interesting to explore means for advanced navigation in our MPM model shape space. How can we help a user, having a target shape in mind, to easily get to this result. This is especially important if the number of input models increases, e.g. when twenty instead of four animals span the shape space.

Another application for our MPM model to investigate is deformable object recognition. It would be interesting to explore, how the system could be used to identify or verify matches of objects, that are in the spanned shape space, but not given in the input examples. For example it should be possible to identify a deer as a four legged animal considering Figure 5.19, given only the four other animals. To do this we have to explore methods for initializing the MPM to possible targets in a scene.

Another direction is to investigate if choosing to partition the shape into discrete, linearly morphing parts is limiting the shape space to hard. There we could examine more general techniques for assembling and combining partial information from example shapes, beyond discrete part segmentations.

References

Adams, B., Ovsjanikov, M., Wand, M., Seidel, H.-P. and Guibas, L. J. 2008. Meshless modeling of deformable shapes and their motion. In *Symposium on Computer Animation.* 115

Alexa, M., Beht, J., Cohen-Or, D., Fleishman, S., Levin, D. and Silva, C. 2003. Computing and rendering point set surfaces. *IEEE Trans. Visualization and Comp. Graphics 9*, 1, 315. 45

Allen, B., Curless, B. and Popović, Z. 2003. The space of human body shapes: reconstruction and parameterization from range scans. In *Proc. ACM SIGGRAPH.* 40, 53, 67

Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J. and Davis, J. 2005. SCAPE: Shape completion and animation of people. *Proc. ACM SIGGRAPH 24*, 3, 408–416. 68

Au, O. K.-C., Tai, C.-L., Cohen-Or, D., Zheng, Y. and Fu, H. 2010. Electors voting for fast automatic shape correspondence. *Computer Graphics Forum 29*, 2, 645–654. 67

Baran, I., Vlasic, D., Grinspun, E. and Popović, J. 2009. Semantic deformation transfer. *ACM Trans. Graph. 28*, 36. 95

Barbic, J., da Silva, M. and Popovic, J. 2009. Deformable object animation using reduced optimal control. *ACM Trans. Graph. 28*, 53:1–53:9. 95

BEN-CHEN, M., WEBER, O. AND GOTSMAN, C. 2009. Spatial deformation transfer. In *Symposium on Computer Animation*. 94

BERNER, A., BOKELOH, M., WAND, M., SCHILLING, A. AND SEIDEL, H.-P. 2008. A graph-based approach to symmetry detection. In *IEEE/EG International Symposium on Volume and Point-Based Graphics*, Eurographics Association, Los Angeles, CA, 1–8.

BERNER, A., BOKELOH, M., WAND, M., SCHILLING, A. AND SEIDEL, H.-P. 2009. Generalized intrinsic symmetry detection. Research Report MPI-I-2009-4-005, Max-Planck-Institut für Informatik, August.

BERNER, A., BURGHARD, O., WAND, M., MITRA, N. J., KLEIN, R. AND SEIDEL, H.-P. 2011. A morphable part model for shape manipulation. Research Report MPI-I-2011-4-005, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, December.

BERNER, A., WAND, M., MITRA, N., MEWES, D. AND SEIDEL, H.-P. 2011. Shape analysis with subspace symmetries. In *Computer Graphics Forum (Proc. Eurographics)*.

BESL, P. J. AND MCKAY, N. 1992. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2, 239–256. 12

BLANZ, V. AND VETTER, T. 1999. A morphable model for the synthesis of 3d faces. In *Proc. SIGGRAPH*. 67, 91, 112

BOKELOH, M., BERNER, A., WAND, M., SEIDEL, H.-P. AND SCHILLING, A. 2008. Slippage features. Tech. rep., Wilhelm Schickard Institut, University of Tbingen. 11, 13

BOKELOH, M., BERNER, A., WAND, M., SEIDEL, H.-P. AND SCHILLING, A. 2009. Symmetry detection using line features. *Computer Graphics Forum (Proc. EUROGRAPHICS 2009) 28*, 2, 697–706. 22, 28, 42, 43, 57, 62

Bokeloh, M., Wand, M. and Seidel, H.-P. 2010. A connection between partial symmetry and inverse procedural modeling. *ACM Trans. Graph. 29* (July), 104:1–104:10. 100, 108

Botsch, M. and Sorkine, O. 2008. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1, 213–230. 94

Botsch, M., Pauly, M., Gross, M. and Kobbelt, L. 2006. Primo: coupled prisms for intuitive surface modeling. In *Symposium on Geometry Processing*, 11–20. 94

Boyd, S. and Vandenberghe, L. 2004. *Convex Optimization.* Cambridge University Press, New York, NY, USA. 104

Bronstein, A. M., Bronstein, M. M. and Kimmel, R. 2006. Generalized multidimensional scaling: A framework for isometry-invariant partial surface matching. *Proceedings of the National Academy of Science (PNAS) 103*, 5, 1168–1172. 40, 53

Brown, B. and Rusinkiewicz, S. 2007. Global non-rigid alignment of 3-d scans. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3, 21. 53, 115

Burghard, O., Berner, A., Wand, M., Mitra, N., Seidel, H. P. and Klein, R. 2013. Compact part-based shape spaces for dense correspondence estimation. In preparation. 106, 107, 118

Chen, Y. and Medioni, G. 1992. Object modelling by registration of multiple range images. *Image Vision Comput. 10*, 3, 145–155. 12

Cootes, T., Edwards, G. and Taylor, C. 2001. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 6, 681–685. 91, 94

Davies, R., Twining, C., Cootes, T., Waterton, J. and Taylor, C. 2002. A minimum description length approach to statistical shape modeling. *IEEE Transactions on Medical Imaging 21*, 5 (May), 525–537. 107

Douros, I. and Buxton, B. 2002. Three-dimensional surface curvature estimation using quadric surface patches. In *Proc. Scanning.* 45

Dryden, I. L. and Mardia, K. 1998. *Statistical shape analysis.* Wiley series in probability and statistics: Probability and statistics. J. Wiley. 69

Felzenszwalb, P. and Huttenlocher, D. 2005. Pictorial structures for object recognition. *Intl. J. Computer Vision 61*, 1, 55–79. 11

Feng, W.-W., Kim, B. and Yu, Y. 2008. Real-time data-driven deformation using kernel canonical correlation analysis. *ACM Transactions on Graphics 27*, 3, 91:1–91:9. 95

Fischler, M. A. and Bolles, R. C. 1987. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision: issues, problems, principles, and paradigms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 726–740. 15

Gal, R. and Cohen-Or, D. 2006. Salient geometric features for partial shape matching and similarity. *ACM Trans. Graph. 25*, 1, 130–150. 10, 11, 40, 67

Gal, R., Shamir, A., Hassner, T., Pauly, M. and Cohen-Or, D. 2007. Surface reconstruction using local shape priors. In *Proc. Symp. Geometry Processing.* 31

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, 33:1–33:10. 90, 95

Gelfand, N. and Guibas, L. J. 2004. Shape segmentation using local slippage analysis. In *Proc. Symp. Geometry processing*, 214–223. 13

Giorgi, D., Biasotti, S. and Paraboschi, L. 2007. Shape retrieval contest 2007: Watertight models track. In *SHREC'07 - Shape Retrieval Contest 2007.* 87, 117

Gumhold, S., Wang, X. and MacLeod, R. 2001. Feature extraction from point clouds. In *Proc. Meshing Roundtable.* 44

Harris, C. and Stephens, M. 1988. A combined corner and edge detection. In *Proc. 4th Alvey Vision Conference*, 147–151. 23

Hasler, N., Stoll, C., Sunkel, M., Rosenhahn, B. and Seidel, H.-P. 2009. A statistical model of human pose and body shape. *Computer Graphics Forum 28*, 2, 337–346. 67, 68, 91, 94

Hilaga, M., Shinagawa, Y., Kohmura, T. and Kunii, T. L. 2001. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proc. ACM SIGGRAPH.* 40, 67

Hildebrandt, K., Polthier, K. and Wardetzky, M. 2005. Smooth feature lines on surface meshes. In *Proc. Symp. Geometry Processing.* 44

Hill, A. and Taylor, C. J. 1994. Automatic landmark generation for point distribution models. In *BMV*, 429–438. 107

Hubo, E., Mertens, T., Haber, T. and Bekaert, P. 2007. Self-similarity-based compression of point clouds, with application to ray tracing. In *SPBG'07*, 129–137. 10

Igarashi, T., Moscovich, T. and Hughes, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Transactions on Computer Graphics 24*, 3, 1134–1141. 90, 94

Kazhdan, M., Chazelle, B., Dobkin, D., Funkhouser, T. and Rusinkiewicz, S. 2003. A reflective symmetry descriptor for 3d models. *Algorithmica 38*, 1, 201–225. 10, 11

Kent, J. T., Mardia, K. V., West, J. M. and Jt, L. L. 1996. Ridge curves and shape analysis. In *In The British Machine Vision Conference*, 43–52. 39

Kilian, M., Mitra, N. J. and Pottmann, H. 2007. Geometric modeling in shape space. *Proc. ACM SIGGRAPH 26*, 3, #64, 1–8. 90

Kirby, M. and Sirovich, L. 1990. Application of the karhunen-loeve procedure for the characterization of human faces. *IEEE PAMI 12*, 1, 103–108. 67

Kkai, I., Finger, J., Smith, R. C., Pawlicki, R. and Vetter, T. 2007. Example-based conceptual styling framework for automotive shapes. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling.* 68

Kotcheff, A. C. and Taylor, C. J. 1998. Automatic construction of eigenshape models by direct optimization. *Medical Image Analysis 2*, 4, 303–314. 107

Kraevoy, V., Sheffer, A., Shamir, A. and Cohen-Or, D. 2008. Non-homogeneous resizing of complex models. *ACM Trans. Graph. 27*, 5, 1–9. 95

Lamdan, Y. and Wolfson, H. J. 1988. Geometric hashing: A general and efficient model-based recognition scheme. In *Proc. Int. Conf. Computer Vision.* 10, 40

Li, X. and Guskov, I. 2005. Multiscale features for approximate alignment of point-based surfaces. In *Symp. Geometry Processing*, 217–226. 13

Lipman, Y., Levin, D. and Cohen-Or, D. 2008. Green coordinates. *ACM Trans. Graph. 27* (August), 78:1–78:10. 94

Loy, G. and Eklundh, J. 2006. Detecting symmetry and symmetric constellations of features. In *Proc. ECCV*, 508–521. 2, 10, 11, 39, 66

Martinet, A., Soler, C., Holzschuch, N. and Sillion, F. 2006. Accurate detection of symmetries in 3d shapes. *ACM Trans. on Graphics 25*, 2, 439 – 464. 10, 11, 40, 67

Mitra, N. J., Guibas, L. J. and Pauly, M. 2006. Partial and approximate symmetry detection for 3d geometry. *Proc. ACM SIGGRAPH 25*, 3, 560–568. 2, 10, 11, 17, 39, 66, 82

Mitra, N. J., Guibas, L. and Pauly, M. 2007. Symmetrization. In *Proc. ACM SIGGRAPH*, vol. 26, 63. 2, 10

Mitra, N. J., Bronstein, A. and Bronstein, M. 2010. Intrinsic regularity detection in 3d geometry. In *Proc. ECCV*, 398–410. 66

Mitra, N. J., Pauly, M., Wand, M. and Ceylan, D. 2012. Symmetry in 3d geometry: Extraction and applications. In *EG 2012 - State of the Art Reports*, 29–51. 2

Ohtake, Y., Belyaev, A. and Seidel, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. In *Proc. ACM SIGGRAPH*, 609–612. 39, 43, 44, 72

Ovsjanikov, M., Sun, J. and Guibas, L. 2008. Global intrinsic symmetries of shapes. In *Proc. Eurographics Symp. on Geometry Processing*, 1341–1348. 40

Pauly, M., Keiser, R. and Gross, M. 2003. Multi-scale feature extraction on point-sampled models. In *Proc. Eurographics*. 44

Pauly, M., Mitra, N., Giesen, J., Gross, M. and Guibas, L. J. 2005. Example-based 3d scan completion. In *Proc. Symp. Geometry Processing*, 23–32. 31

Pauly, M., Mitra, N. J., Wallner, J., Pottmann, H. and Guibas, L. 2008. Discovering structural regularity in 3D geometry. *Proc. ACM SIGGRAPH 27*, 3, #43, 1–11. 2, 31, 39, 66, 82

PODOLAK, J., SHILANE, P., GOLOVINSKIY, A., RUSINKIEWICZ, S. AND FUNKHOUSER, T. 2006. A planar-reflective symmetry transform for 3D shapes. *Proc. ACM SIGGRAPH 25*, 3, 549–559. 2, 10, 11, 39, 66

RAVIV, D., BRONSTEIN, A. M., BRONSTEIN, M. M. AND KIMMEL, R. 2007. Symmetries of non-rigid shapes. In *Proc. Workshop on Non-rigid Registration and Tracking through Learning.* 40

SCHNABEL, R., WAHL, R. AND KLEIN, R. 2007. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (June), 214–226. 88

SCHNABEL, R., WESSEL, R., WAHL, R. AND KLEIN, R. 2008. Shape recognition in 3d point-clouds. In *Proc. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision.* 11, 40, 67

SCHÖLKOPF, B., SMOLA, A. AND MLLER, K.-R. 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation 10*, 5 (July 1), 1299–1319. 67

SEDERBERG, T. W. AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proc. Siggraph*, 151–160. 94

SIMARI, P., KALOGERAKIS, E. AND SINGH, K. 2006. Folding meshes: hierarchical mesh segmentation based on planar symmetry. In *Proc. Eurographics Symp. on Geometry Processing*, 111–119. 10, 11, 40, 67

SORKINE, O. AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 109–116. 90, 94, 99, 103, 114

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C. AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, ACM, New York, NY, USA, 175–184. 90, 94

Sumner, R. W., Zwicker, M., Gotsman, C. and Popović, J. 2005. Mesh-based inverse kinematics. *Proc. ACM SIGGRAPH 24*, 3, 488–495. 68, 94, 115

Tena, J. R., De la Torre, F. and Matthews, I. 2011. Interactive region-based linear 3d face models. In *Proc. ACM SIGGRAPH.* 95

Tevs, A., Bokeloh, M., Wand, M., Schilling, A. and Seidel, H.-P. 2009. Isometric registration of ambiguous and partial data. In *Proc. IEEE CVPR.* 41

Tevs, A., Berner, A., Wand, M., Ihrke, I. and Seidel, H.-P. 2011. Intrinsic Shape Matching by Planned Landmark Sampling. In *Computer Graphics Forum (Proc. EUROGRAPHICS)*, Blackwell, Llandudno, UK, O. Deussen and M. Chen, Eds., Eurographics, 543–552.

Tevs, A., Berner, A., Wand, M., Ihrke, I., Bokeloh, M., Kerber, J. and Seidel, H.-P. 2012. Animation cartography—intrinsic reconstruction of shape and motion. *ACM Trans. Graph. 31*, 2, 1–15.

Thrun, S. and Wegbreit, B. 2005. Shape from symmetry. In *Proc. Int. Conf. Computer Vision.* 11, 31

Turk, M. and Pentland, A. 1991. Face recognition using eigenfaces. In *Proc. IEEE CVPR*, 586–591. 67

Wand, M., Berner, A., Bokeloh, M., Fleck, A., Hoffmann, M., Jenke, P., Maier, B., Staneker, D. and Schilling, A. 2007. Interactive editing of large point clouds. In *Symposium on Point-Based Graphics 2007 : Eurographics / IEEE VGTC Symposium Proceedings*, Eurographics Association, Prague, Czech Republik, B. Chen, M. Zwicker, M. Botsch, and R. Pajarola, Eds., 37–46.

Wand, M., Berner, A., Bokeloh, M., Jenke, P., Fleck, A., Hoffmann, M., Maier, B., Staneker, D., Schilling, A. and Seidel, H.-P. 2008. Processing and interactive editing of huge point clouds from 3d scanners. *Comput. Graph. 32*, 2, 204–220.

WAND, M., ADAMS, B., OVSJANIKOV, M., BERNER, A., BOKELOH, M., JENKE, P., GUIBAS, L., SEIDEL, H.-P. AND SCHILLING, A. 2009. Efficient reconstruction of nonrigid shape and motion from real-time 3d scanner data. *ACM Trans. Graph. 28*, 2, 1–15.

WANG, Y., XU, K., LI, J., ZHANG, H., SHAMIR, A., LIU, L., CHENG, Z. AND XIONG, Y. 2011. Symmetry hierarchy of man-made objects. In *Eurographics 2011*, vol. 30, 287–296. 95

WINKLER, T., DRIESEBERG, J., ALEXA, M. AND HORMANN, K. 2010. Multi-scale geometry interpolation. *CGF (EUROGRAPHICS) 29*, 2, 309–318. 90

XU, K., ZHANG, H., TAGLIASACCHI, A., LIU, L., LI, G., MENG, M. AND XIONG, Y. 2009. Partial intrinsic reflectional symmetry of 3d shapes. *ACM Transactions on Graphics, (Proceedings SIGGRAPH Asia 2009) 28*, 5, 138:1–138:10. 95

ZHANG, L., SNAVELY, N., CURLESS, B. AND SEITZ, S. M. 2004. Spacetime faces: High-resolution capture for modeling and animation. In *Proc. ACM SIGGRAPH*, 548–558. 95

ZHANG, H., SHEFFER, A., COHEN-OR, D., ZHOU, Q., VAN KAICK, O. AND TAGLIASACCHI, A. 2008. Deformation-driven shape correspondence. *Computer Graphics Forum (Proc. SGP) 27*, 5, 1431–1439. 40, 67

ZHENG, Q., SHARF, A., TAGLIASACCHI, A., CHEN, B., ZHANG, H., SHEFFER, A. AND COHEN-OR, D. 2010. Consensus skeleton for non-rigid space-time registration. *Computer Graphics Forum 29*, 2, 635–644. 67

ZHENG, Y., FU, H., COHEN-OR, D., AU, O. K.-C. AND TAI, C.-L. 2011. Component-wise controllers for structure-preserving shape manipulation. vol. 30, 563–572. 90, 95