

A Graph-Based Approach to Symmetry Detection

A. Berner¹, M. Bokeloh¹, M. Wand^{2,3}, A. Schilling¹, H.-P. Seidel³
¹University of Tübingen ²Saarland University ³Max-Panck-Institut Informatik

Abstract

Symmetry detection aims at discovering redundancy in the form of reoccurring structures in geometric objects. In this paper, we present a new symmetry detection algorithm for geometry represented as point clouds that is based on analyzing a graph of surface features. We combine a general feature detection scheme with a RANSAC-based randomized subgraph searching algorithm in order to reliably detect reoccurring patterns of locally unique structures. A subsequent segmentation step based on a simultaneous region growing variant of the ICP algorithm is applied to verify that the actual point cloud data supports the pattern detected in the feature graphs. We apply our algorithm to synthetic and real-world 3D scanner data sets, demonstrating robust symmetry detection results in the presence of scanning artifacts and noise. The modular and flexible nature of the graph-based detection scheme allows for easy generalizations of the algorithm, which we demonstrate by applying the same technique to other data modalities such as images or triangle meshes.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics] Computational Geometry and Object Modeling, I.5.3 [Pattern Recognition] Clustering, I.2.10 [Artificial Intelligence] Vision and Scene Understanding

1. Introduction

Many real-world objects consist of parts that are approximately similar to each other. For example, a façade of a building might contain a number of windows with comparable geometry. The automatic detection of such redundancies is a recent, very interesting research direction [TW05, PSG*06, MGP06, MSH*06, SKS06, MGP07]. Information about similarities within one and the same shape is useful for a variety of applications, such as compression, pattern recognition, shape completion, and statistical noise removal.

In this paper, we propose a new method for detecting reoccurring parts (from now on just called “*symmetries*”) in point clouds from 3D scanning devices. Our paper makes two main contributions: First, we present a novel approach to solving this problem based on building graphs of salient features and an efficient subgraph matching technique. We introduce a new randomized geometric graph matching algorithm, which is combined with a generalized feature detector. We expect that this algorithmic approach might be useful for solving other geometric correspondence problems as well. Second, we explicitly design our algorithm for detection of symmetries in scanned point cloud data sets. Our method handles real-world raw 3D scanner point clouds with noise artifacts robustly.

We evaluate our algorithm on multiple synthetic and real-world 3D scanner data sets. In addition, we also consider generalizations to image data and clean triangle meshes, demonstrating the generality of the novel approach.

2. Related Work

Symmetry detection in 3D shapes has recently gained a lot of interest. 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. [MGP06] propose this type of algorithm, using principal curvature directions to create

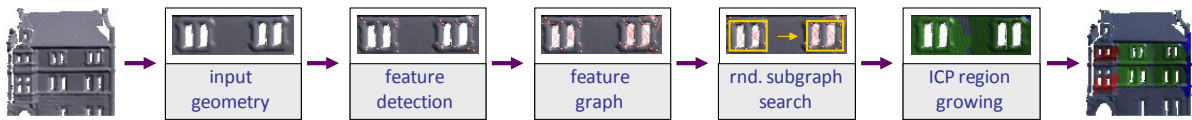


Figure 1: Processing pipeline – first a set of locally unique features is detected, which then form a graph on the object surface. Next, subgraphs with matching topology and approximately matching geometric embedding are extracted. From these discrete candidate matches, data points are assigned to symmetric regions using an ICP-based simultaneous region growing algorithm, which yields the final result.

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 [MGP07]. Another voting approach has been proposed concurrently by [PSG*06], who use voting on reflective planes to detect planar reflective symmetries. Gal and Cohen-Or [GC06] form clusters of quadratic surface patches and use geometric hashing [LW88] to find symmetries in objects. In image data, a Hough transform of salient feature points has been used by Loy and Eklundh [LE06] to find symmetric configurations. Martinet et al. [MSH*06] 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 [SKS06]: 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. [HMH*07] 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. [KCD03] 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 [TW05] 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 [MGP06, GC06, PSG*06, LE06], our algorithm is based on a graph matching strategy. The main advantage of this approach is that it can be potentially 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 [KCD03, MSH*06, SKS06] 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 [FH05] 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. [SWW*08]: 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. To the best of our knowledge, no previous technique has applied graph-based matching techniques to detecting symmetries in geometric objects.

3. 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 recently proposed slippage feature algorithm [BBW*08], which is capable of 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 4. Given this graph, we employ a randomized subgraph search algorithm in order to detect reoccurring patterns in this graph (Section 5). 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 [BM92, CM92] that

performs simultaneous matching and region growing over all detected patterns (Section 6). Generalizations of this basic strategy are discussed in Section 7. Finally, we evaluate our algorithm on various test data sets (Section 8) and conclude with some ideas for further generalizations that the graph-based approach allows for in Section 9.

4. Building the Feature Graph

4.1 Feature Detection

Our whole approach is based on matching graphs of feature points. Therefore, the feature detection step 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 employing the recently proposed “slippage features” detection algorithm that can detect features of arbitrary type with the same reliability as previous state of the art techniques (such as [LG05]). In the following, we give a brief overview of this algorithm; for further details and a quantitative evaluation, see [BBW*08].

Feature detection: The feature detector works directly on point clouds. For each input point \mathbf{p}_i , it analyses a spherical neighborhood $N_\varepsilon^{(i)}(\mathbf{p}_i)$. The key idea is: in order for \mathbf{p}_i to be a good feature point, the auto-alignment problem of $N_\varepsilon^{(i)}(\mathbf{p}_i)$ 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 [GG04]: We 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 [GG04]. We compute a slippage value for all surface points and perform 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, we extend this algorithm by performing the extraction for several neighborhood sizes $\varepsilon_1, \dots, \varepsilon_k$, doubling in each step: $\varepsilon_{i+1} = 2\varepsilon_i$. 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 feature points as \mathbf{k}_i , $i = 1 \dots n$. The scale is given by $\varepsilon(\mathbf{k}_i)$ and the associated neighborhood of input points by $N(\mathbf{k}_i)$.

Feature matching: In order to detect matching features, we use a descriptor based on curvature histograms in concentric rings within each $N(\mathbf{k}_i)$. We consider all pairs of feature points $(\mathbf{k}_i, \mathbf{k}_j)$; correspondence pairs for which the descriptors do not match are discarded. The remaining pairs of features are candidate matches where the local neighborhood of the feature point is approximately similar.

4.1.1 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 reoccurring 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, reoccurring 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 $G = (K, E)$, $K = \{k_1, \dots, k_n\}$, $E \subseteq \{1, n\} \times \{1, n\}$.

We also store the set of matching feature pairs along with the matching residuals as a quality measure. Please note that no edges are added to the graph at this stage; the feature graph encodes only the spatial relation of surface features.

5. Sub-Graph Matching

5.1 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* $S^{(i)} := \{S_1^{(i)}, \dots, S_{\#S^{(i)}}^{(i)}\}$ 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 $dist_G$:

$$\forall i \in \{1..n_S\} : \forall j_1, j_2 \in \{1..\#S^{(i)}\} : dist_G(S_{j_1}^{(i)}, S_{j_2}^{(i)}) \leq \varepsilon$$

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, bricks), but these instances have to consist of disjoint features \mathbf{k} . 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 [FB87] to maximize the descriptive power of the extracted instances.

5.2 The Subgraph Distance Function

As distance function, we currently use a simple rigid matching function. We consider this a first step in exploring graph-based symmetry detection. It is at least conceptually straightforward to apply more general matching criteria such as a graph matching with isometric rather than Euclidian embedding or a fault tolerant comparison of subgraph topologies.

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 5.5). 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 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 .

5.3 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 disjointness 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 one and 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

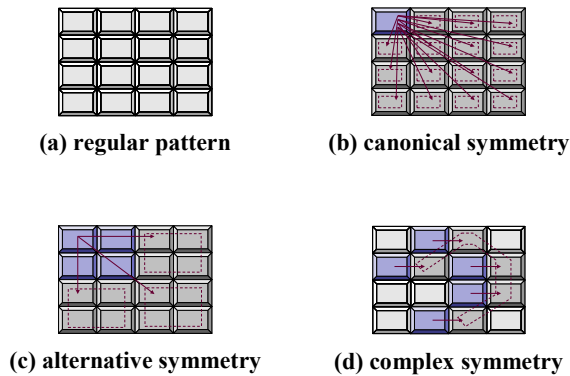


Figure 2: A simple regular pattern can exhibit a large number of maximal symmetries.

another feature that is reoccurring 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.

5.4 Inherent Ambiguities

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). 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 reoccur one or more times. Such regular structures can be described using group theory [MGP06]: We consider the group of rigid transformations. The transformations \mathbf{T} that map regular patterns to a symmetric configuration such as depicted in Figure 2 can be described by repeated application of generator transformations \mathbf{G}_i :

$$\mathbf{T}_{i_1, \dots, i_k} = \mathbf{G}_1^{i_1} \circ \mathbf{G}_2^{i_2} \circ \dots \circ \mathbf{G}_k^{i_k}$$

where the range of exponents i_j is subject to additional index constraints (in our example, the instance is only valid for $i_1, i_2 \in \{0, \dots, 3\}$ 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.

5.5 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 $(\varepsilon - dist(e_i, e_j))$ to each edge correspondence and add up the score for all reoccurrences of each edge, thus favoring edges that reoccur more frequently. We then use im-

portance sampling according to this importance value to randomly create an initial symmetry set that contains only one instance of one single edge reoccurring 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).

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 reoccurring subgraphs found.

Robustness to noise: Our method relies on distinguishable descriptors. If a dataset 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 3a right).

6. ICP-based Region Growing

Having found symmetric features constellations as described in the previous section, we now want to transfer these 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, \dots, F_m , and rigid transformations $\mathbf{T}_{i,j}$ that maps 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 $\mathbf{k} \in F_1$ 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 \mathbf{x} , we compute a local tangent coordinate system via principal component analysis (PCA), fit a bivariate quadratic polynomial to the data points in a least squares sense and move the point \mathbf{x} onto the computed polynomial. We use a standard Gaussian window function $\omega = \exp(-\|\mathbf{x}\|^2/\sigma^2)$ 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. Using this surface representation, we also compute a normal for the projected point. In order to decide whether points \mathbf{x} and \mathbf{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 algo-

rithm 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 T_{ij} 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.

7. Extensions

The 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 [HS88] 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.

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.

8. Results and Applications

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 3(a) 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 chapter 5.5.

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 3b). 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. The second example shows a scan of the “Zwinger” in Dresden, a historical building from the 18th century Figure 3(c). 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 3d). In 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.

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 3(e). Application of our algorithm to bitmap images that show reoccurring subimages also leads to a very good recognition rate. For example, we were able to fully automatically identify the symbols of a circuit diagram and identify reoccurring phrases in a Japanese translation of the traditional German poem “ode to joy” (Figure 3f).

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. It depends 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.

9. Conclusion and Future Work

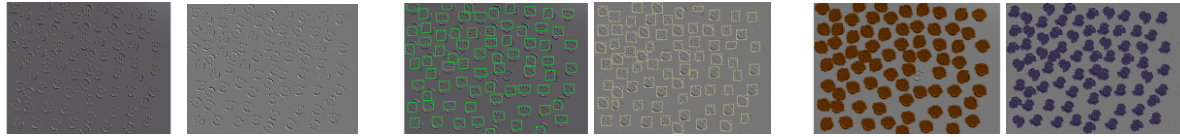
We have presented a novel approach to symmetry detection in geometric data. Unlike previous approaches, the new technique is based on a subgraph matching algorithm. The different approach has some advantages over previous techniques: It does not require the detection of the global symmetry structure prior to handling small scale symmetries and can be potentially generalized to more general matching criteria that cannot be described by voting in transformation spaces. In this paper, we deliver a 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. In future work, we would like to examine generalizations to isometric matching (using geodesic lengths on surfaces instead of rigid Euclidian transformations) as validation criterion, as well as more general, topology-based graph matching techniques. The graph-based approach might also be useful in more general shape matching problems, such as multi-view deformable registration.

Acknowledgements

The authors wish to thank Markus Wacker, Allan Chalmers and Simon Pabst for providing data sets and the anonymous reviewers for their helpful comments. This work has been supported by DFG grant “Perceptual Graphics”, the Max Planck Center VCC and Cluster of Excellence “Multi-Modal Computation and Interaction” at Saarland University.

References

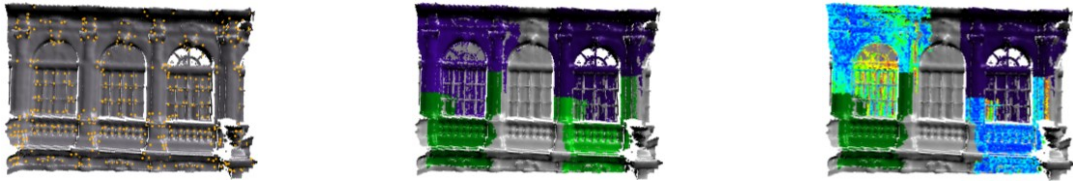
- [BBW*08] BOKELOH, M., BERNER, A., WAND, M., SEIDEL, H.-P., SCHILLING, A.: Slippage features. Technical Report, WSI-2008-03, University of Tübingen, 2008
- [BM92] BESL, P. J., MCKAY, N.: A Method for Registration of 3-D Shapes. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14, 239-256, 1992.
- [CM92] CHEN, Y., MEDIONI, G.: Object modelling by registration of multiple range images. In: *Image Vision Comput.*, 10, 145–155, 1992.
- [FH05] FELZENSZWALB, P., HUTTENLOCHER, D.P.: Pictorial Structures for Object Recognition. In: *Intl. J. Computer Vision*, 61(1), 55–79, 2005.
- [FB87] FISCHLER, M. A., BOLLES, R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Morgan Kaufmann Publishers Inc.*, 1987.
- [GC06] GAL, R., COHEN-OR, D. Salient geometric features for partial shape matching and similarity. In: *ACM Trans. Graph.* 25(1), 130-150, 2006.
- [GG04] GELFAND, N., GUIBAS, L. J.: Shape segmentation using local slippage analysis. In: Proc. Symp. Geometry Processing (SGP04), 214–223, 2004.
- [HS88] HARRIS, C., STEPHENS, M.: A Combined Corner and Edge Detection. In: Proceedings of The Fourth Alvey Vision Conference, 147–151, 1988.
- [HMH*07] HUBO, E., MERTENS, T., HABER, T., BEKAERT, P.: Self Similarity-Based Compression of Point Clouds, with Application to Ray Tracing. In: *Proc. Symp. Point-Based Graphics*, 2007.
- [KCD03] KAZHDAN, M., CHAZELLE, B., DOBKIN, D., FUNKHOUSER, T., RUSINKIEWICZ, S.: A Reflective Symmetry Descriptor for 3D Models. In: *Algorithmica*, 38, 201–225, Springer, 2003.
- [LW88] Lamdan, Y., Wolfson, H. J.: Geometric hashing: A general and efficient model-based recognition scheme. In: Int. Conf. Computer Vision (ICCV'88), 238–249, 1988.
- [LG05] LI, X., GUSKOV, I.: Multiscale Features for Approximate Alignment of Point-based Surfaces. In: *Proc. Symp. Geometry Processing (SGP05)*, 217–226, 2005.
- [LE06] LOY, G., EKLUNDH, J.O.: Detecting Symmetry and Symmetric Constellations of Features. ECCV (2) 2006: 508-521
- [MSH*06] MARTINET, A., SOLER, C., HOLZSCHUCH, N., SILLION, F.: Accurate Detection of Symmetries in 3D Shapes. In: *ACM Transactions on Graphics*, 25(2), 439 – 464, 2006.
- [MGP06] MITRA, N. J., GUIBAS, L. J., PAULY, M.: Partial and approximate symmetry detection for 3D geometry. In: *ACM Trans. Graph.*, 25(3), 560-568, 2006.
- [MGP07] MITRA, N. J., GUIBAS, L. & PAULY, M.: Symmetrization. In: *ACM Trans. on Graphics* 26(3), 2007.
- [PSG*06] PODOLAK, J., SHILANE, P., GOLOVINSKIY, A., RUSINKIEWICZ, S., FUNKHOUSER, T.: A Planar-Reflective Symmetry Transform for 3D Shapes. In: *ACM Trans. on Graphics* 25(3), 2006.
- [SWW*08] Schnabel, R., Wessel, R., Wahl, R., Klein, R.: Shape Recognition in 3D Point-Clouds. In: *Proc. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2008.
- [SKS06] SIMARI, P., KALOGERAKIS, E., SINGH, K.: Folding meshes: hierarchical mesh segmentation based on planar symmetry. In: *Proc. Symp. Geometry Processing (SGP06)*, 111–119, 2006.
- [TW05] THRUN, S., WEGBREIT, B.: Shape from Symmetry. In: *Proc. Int. Conf. Computer Vision (ICCV '05)*, 1824–1831, 2005.



(a) Faces – synthetic data set: left: features, middle: detected subgraphs, right: symmetry detection result. The left image in each column shows the plain heightfield and the right image the version with $\sigma = \pm 10\%$ Gaussian noise.



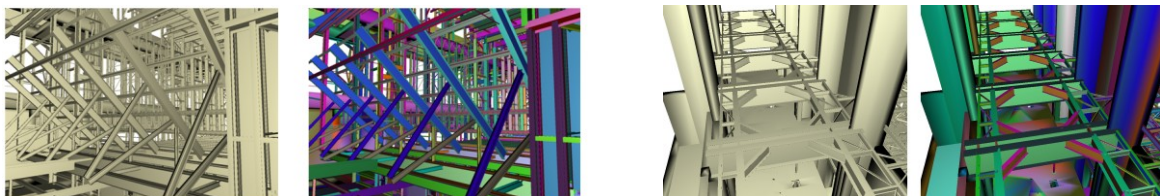
(b) Engraved horseman (historical artifact): Left: Slippage features, middle: symmetry detection result, right: matching residuals (blue = low, red = high). Data set courtesy of Allan Chalmers.



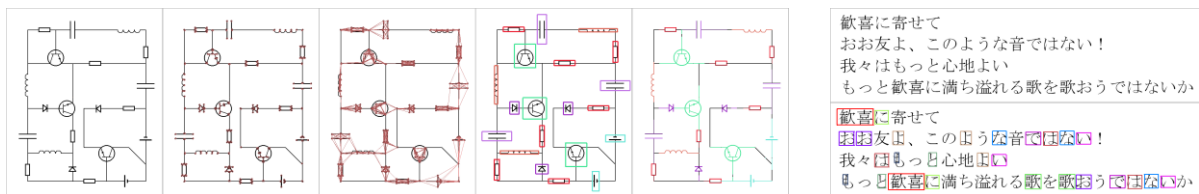
(c) Zwinger at Dresden (3D scan of the actual building): left: features, middle: symmetry detection result, right: matching residual. Data set courtesy of Markus Wacker.



(d) Clay house model: Left detected features, middle: symmetric parts (instances are color coded), right: matching residuals.



(e) 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.



(g) Application to image data: Left – circuit diagram (from left to right: input, features, feature graph, detected subgraphs, final symmetries after growing; corresponding instance obtain the same color). Right – Japanese text; top: input image, bottom: Recognized subgraphs.

Figure 3: Example data sets with symmetry detection results.