EUROGRAPHICS 2009 / P. Dutré and M. Stamminger
(Guest Editors)

Volume 28 (2009), Number 2

Symmetry Detection Using Feature Lines

M. Bokeloh[1] and A. Berner[1] and M.Wand[2,3] and H.-P. Seidel[3] and A. Schilling[1]

[1]WSI/GRIS, University of Tuebingen, Germany
[2]Saarland University
[3]Max-Planck Institute Informatik

Abstract

In this paper, we describe a new algorithm for detecting structural redundancy in geometric data sets. Our algorithm computes rigid symmetries, i.e., subsets of a surface model that reoccur several times within the model differing only by translation, rotation or mirroring. Our algorithm is based on matching locally coherent constellations of feature lines on the object surfaces. In comparison to previous work, the new algorithm is able to detect a large number of symmetric parts without restrictions to regular patterns or nested hierarchies. In addition, working on relevant features only leads to a strong reduction in memory and processing costs such that very large data sets can be handled. We apply the algorithm to a number of real world 3D scanner data sets, demonstrating high recognition rates for general patterns of symmetry.

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

1. Introduction

Symmetry detection is a recent research direction in geometry processing: The task of a symmetry detection algorithm is to decompose the input geometry into a set of parts, and a set of mapping functions that reassemble an approximation of the original object from these parts up to a desired level of accuracy. The ability to compute such decompositions automatically from general input geometry is an important tool for many computer graphics applications. For example, a symmetry decomposition provides a natural way of compressing geometry, it allows for improving the quality of partially and noisily scanned objects, and also facilitates "intelligent" editing of geometry, such as propagating edits to multiple instances of one and the same building block.

Figure 1: *Line features for the "old town hall" data set.*

Our paper addresses the problem of finding rigid symmetries. This means we are looking for parts in the model that can be reused under orthogonal mappings (rotation and mirroring) and translations to assemble the input model more compactly. The most successful techniques for detecting such types of symmetry are based on transformation voting, where transformations between candidate pairs of potentially corresponding points are inserted into a Hough-Transform space to vote for dominant transformations [MGP06, LE06, PSG*06, MGP07, PMW*08]. This approach is very elegant and gives good results in many cases. The main limitation is however that all transformations end up in the same voting space with spatial correlation information lost so that the problem of isolating the relevant symmetries becomes harder when the number of different symmetries grows [PMW*08]. As a consequence, these techniques have so far only been applicable for coarse, large scale symmetries [MGP06, PSG*06], in special cases were

the model can be decomposed hierarchically [MGP06], or if the symmetries form regular patterns in transformation space [PMW*08].

The goal of this work is to detect a large number of symmetries without such prerequisites. The key observation for an improvement in recognition capabilities is that we can expect some spatial coherence: The instantiated part, which is found several times in the model under a symmetry transformation, will typically itself form a localized, coherent spatial object: Nearby points are likely to show the same symmetry pattern. This information is lost in transformation space.

In order to exploit this information, we choose a different approach: We first identify discrete features on the object surface and form a spatial neighborhood graph of such features. Next, we examine the graph for reoccurring patterns using a randomized matching algorithm. Finally, the results from matching local clusters of features are transferred and validated on the original geometry. Matching of locally coherent feature sets for symmetry detection has been examined previously by Berner et al. [BBW*08]. However, their sets of stable keypoints are often not dense enough to find large sets of symmetries. We improve upon their algorithm by using pairs of linearly slippable regions instead of keypoints and a different matching algorithm that performs multiple nested loops of RANSAC-based pattern matching [FB81]. The new strategy yields both a substantial improvement in recognition rates as well as a significant improvement in running times. As an example application for our technique, we apply the algorithm to symmetry-based automatic reconstructing of scanning data [GSH*07, PMW*08].

Our paper makes two main contributions: First, it proposes a novel symmetry detection algorithm that can find rigid symmetries in general configurations. The algorithm works efficiently on large data sets, up to two orders of magnitude more complex than previous state-of-the art results. Second, we develop a framework for rapid geometry matching based on sparse sets of feature lines. In addition, we apply the detected symmetries to automatic quality improvement in scanner data and show that the large amount of redundancy discovered can substantially improve the model quality in a fully unsupervised way.

2. Related Work

Transformation voting techniques for symmetry detection in geometry have been introduced by Mitra et al. [MGP06], and Podolak et al. [PSG*06], as well as Loy and Eklundh [LE06] for image features. While the two latter consider reflections and rotational symmetries, respectively, the technique of Mitra et al. is slightly more general than ours by also allowing for scaling. Although it would be conceivable to extend our technique to scaling by considering triples instead of pairs of line features (see Section 5), we restrict ourselves to the rigid case for simplicity. The symmetry detection technique of Mitra et al. has subsequently been extended to automatic object symmetrization [MGP07]. In recent work [PMW*08], an extension to find regular patterns has been proposed: If the objects show regular patterns of symmetries, such as regularly spaced rows of windows in a building, one can explicitly look for these patterns in transformation space to obtain much more stable results. However, this approach can only identify such patterns; symmetric parts in isolated instances or as members of different patterns are not identified as belonging to the same class of objects.

Gal and Cohen-Or [GCO06] propose a variant of transformation voting that uses geometric hashing [LW88] of salient features. The authors give examples of detecting a small number of symmetric parts within an object. Simari et al. [SKS06] detect planar reflective symmetries by computing an auto-alignment of parts of a shape with itself using iteratively reweighted least-squares. This yields a nested symmetry decomposition. Such approaches are limited to cases where large and small scale symmetry patterns correlate. Martinet et al. [MSHS06] propose a technique that uses a transformation to generalized moment functions in order to compute global symmetries of 3D shapes. Kazhdan et al. [KCD*03] analyze objects for central symmetry and use this as a descriptor for shape retrieval. A very interesting application of symmetry detection is shape completion: Thrun and Wegbreit [TW05] compute symmetries of partially scanned objects to complement the partially shape. Schnabel et al. [SWWK08] use graphs of fitted geometry primitives to perform object recognition.

Berner et al. [BBW*08] perform subgraph matching in graphs of feature points, which are defined as regions in which the auto-alignment problem of local patches is fully constrained. We relax this requirement to partial constraints (feature lines), which can be detected more easily and frequently, thus improving the results drastically. Our method needs a feature detection step that extracts surface lines [GWM01, PKG03, OBS04, HPW05]. In principle, any such feature line detection algorithm can be used at this stage. The specific method presented in this paper has the advantage that it outputs the curvature of the lines as side information without additional computations, which we use to detect stable bases and to prune line matches. This information is retrieved using slippage analysis, which yields stable results even in case of complex surface pattern with rather vague curvature information. Our method is based on local surface fitting [OBS04] so that we obtain detailed and stable results with higher resolution than previous point-based techniques based on a local planarity analysis via PCA [GWM01, PKG03]. The point-sample-based representation of our feature lines furthermore facilitates handling of complex line patterns in the context of our application, where the knowledge of line topology is not relevant.

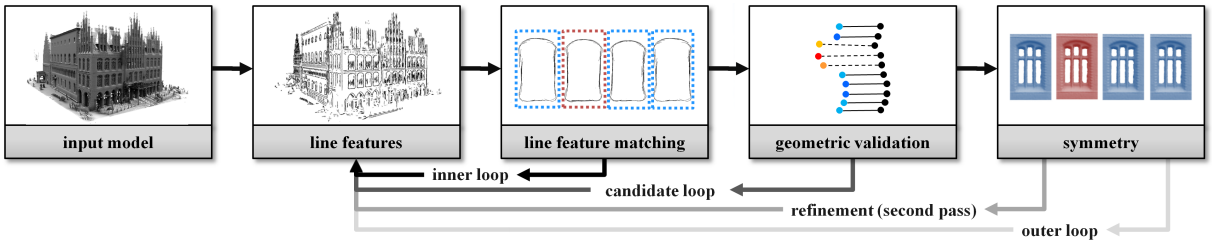

Figure 2: *Overview: We extract line features from the input model. Inside the inner loop, we search for symmetric constellations of line features and pass the best candidate to the validation stage, where we verify it taking all data points into account. We repeat this process several times inside the candidate loop and return the best candidate (one symmetric part with its instances). This set is then refined, in order to find more instances. An outer loop repeats the process to find several symmetries.*

3. Algorithm Overview

Our algorithm consists of three main stages (see Figure 2): *Feature detection* (Section 4), *line-feature matching* (Section 5) and *geometric validation* (Section 6). The last two steps are used iteratively in an outer loop (Section 7). Finally, the symmetry information is then used for scan reconstruction and other applications (Section 8).

The first stage extracts line features and identifies suitable pairs of these features to define local coordinate systems that we call *"bases"*. The second stage then tries to find subsets of such bases so that line features in their local neighborhood match. In order to verify the matching, the algorithm uses an *"iterative-closest-line (ICL)"* algorithm that aligns and compares sets of line features. The rationale behind this design is that a comparison of a sparse set of line features is substantially more efficient than comparing actual geometry, while still achieving a very high matching accuracy. Probably even more important, matching of line features is more resistant to holes and outliers in the data than conventional geometry alignment techniques such as ICP [CM92, BM92] (see Figure 4).

In the next stage of the algorithm, geometric validation, we transfer our findings to actual geometry, i.e. the input point cloud. As the line feature representation is sparse, this step is necessary in order to correctly assign geometric regions to found instances. It computes disjoint instances of geometry that are replicated by the symmetry transformations. Line feature matching and geometric validation are repeated in a "candidate (RANSAC) loop", outputting the best match only, to avoid spurious matching results. Once we know the actual instances of symmetric geometry, we refine our results in a second matching pass to increase the number of recognized instances: We detect additional bases belonging to the instances by checking for overlap with the points computed in the geometric validation stage. In addition, we also try to extrapolate known transformations to discover possibly undetected parts in a transformation prediction step. Afterwards, geometric validation is performed again. This yields a single symmetric part with all its final, typically larger, set of instances. Finally, the outer loop of the algorithm iterates this whole procedure several times in order to find several different classes of symmetry in the model.

4. Feature Extraction

The task of the feature extraction step is to discretize the continuous matching problem, which allows us to perform a discrete matching of feature pattern rather than continuous matching techniques. The crucial point at this stage is to retain the important information about object shape when reducing the object from a continuous surface to a finite set of features. As we are looking for rigid mappings, the local feature geometry must determine a rigid mapping. This means, the associated geometry must "lock-in"; in other words, the auto-alignment problem of registering this geometry with itself must be constrained in all 6 degrees of freedom.

This problem can be addressed by a *slippage analysis* of the local geometry [GG04]. Slippage analysis determines whether the problem of aligning a piece of geometry with itself is well posed: It sets up an ICP objective function (the sum of all point-to-plane surface distances of surface points with themselves), with 6 degrees of freedom (3 translational and 3 rotational). The eigenvalues of the Hessian of the objective function determine whether the geometry is *slippable*: Any surface is necessarily constrained in at least 3 degrees of freedom when being matched to itself. The eigenvectors of the Hessian with eigenvalues (numerically close to) zero, here called *slippage vectors*, describe the associated degrees of freedom. The slippage vectors may mix translational and rotational degrees of freedom (see the paper of Gelfand and Guibas for details [GG04]). Berner et al. [BBW*08] use this information to pick points that are maximally constrained in all degrees of freedom and maximize this property in scale space. However, the number of such points that can be identified stably is limited and becomes the main bottleneck for the purpose of symmetry detection.

In order to improve the recognition performance, we need

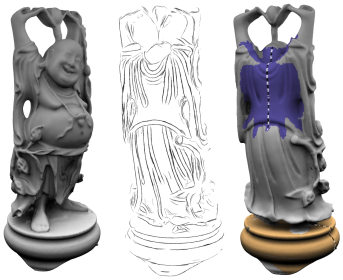

Figure 3: *Line features for the Happy Buddha model (middle) and two detected symmetries (right).*

to generalize the notion of a feature. The main idea of our feature detection strategy is to not rely on a single feature to define correspondences but employ groups of more than one feature. If we have feature regions that are completely unslippable, a single such region yields a rigid constraint. For a slippable region with one degree of freedom (one dimensional null space of the Hessian), we need two regions and have to demand from their slippability vectors (eigenvectors of the Hessian with eigenvalue zero) to span different subspaces. Accordingly, we can also look for feature regions with two or three slippable degrees of freedom. Depending on the dimension of the intersection of their nullspaces, we might need at least two or three such regions to fix a rigid mapping.

In this paper, we only consider the case of pairs of feature regions that are still slippable in one dimension, with a non-zero translational component. This corresponds to regions with arbitrary cross section extruded along (locally) straight or circular lines. As such regions will have a one-dimensional symmetry direction, we refer to them as *line features*. We do not yet consider the more general case of 2D or 3D slippable regions. In practice, 1D slippable feature lines contain most of the salient information and thus provide a both compact and efficient representation. They work particularly well for man-made objects such as architectural models (Figure 1), but we also obtain descriptive results for many natural objects such as faces or sculptures (Figure 3). Our goal is now to extract such 1D-slippable regions from the input geometry. As being slippable in one direction, these regions will have a linear structure.

MLS Line Features: Conceptually, we base our line feature detection on a moving least squares scheme: We define a projection operator that moves points orthogonal to the local 1D slippage direction and the surface normal, trying to maximize curvature along this line. Performing such projections repeatedly will yield a point-sampled feature line representation.

As input we expect a point cloud $\tilde{\mathcal{S}} = \{\mathbf{x}_1 ... \mathbf{x}_n\}$ sampled from a smooth manifold $\mathcal{S} \subseteq \mathbb{R}^3$ where each point $\mathbf{x}_i$ is equipped with a normal vector $\mathbf{n}_i$. We now define the projection operator: In order to project a point $\mathbf{p}$, we first compute a local slippage analysis of $\mathbf{p}$ (we use a Gaussian window function centered at $\mathbf{p}$ with standard deviation ε_{feat}). Slippage analysis gives us a set of unit-norm eigenvectors $\mathbf{v}_1, ..., \mathbf{v}_6$ that represents slippable motions and the corresponding eigenvalues $\lambda_1 \leq \ldots \leq \lambda_6$. We are only interested in the first eigenvector $\mathbf{v}_1$ that represents the most slippable rigid motion. From $\mathbf{v}_1$ we extract the translation part $\mathbf{s}$, which becomes our estimate of the tangential direction of the line. We now find a local maximum of the mean curvature along the direction $\mathbf{s} \times \mathbf{n}$. We use a simple gradient descent algorithm that stops were the gradient vanishes. To compute the mean curvature of a given surface patch we fit a local quadratic patch in a least-squares sense, again using the same Gaussian window function, and compute the eigenvalues of the quadratic component. We also recompute the slippage analysis at the final feature point and store the first eigenvector.

The Gaussian windowing function in the MLS scheme limits the frequency resolution of the line detection. Therefore, we only need to consider a sparse sampling of the feature lines: We compute a Poisson disc sampling of the point cloud with radius ε_{feat} and project only this subset. For sample points where the local geometry is slippable in more than one direction (we used $\lambda_1/\lambda_2 > 0.5$ in every example), we stop the projection immediately and reject the point. We also reject points that move by more than ε_{feat} (they will be handled by neighboring samples) and points with less than 40 neighboring points in the Gaussian window.

Our algorithm outputs a sparse, point-based representation of feature lines, along with the local tangential direction and curvature of the feature line, which are computed from the translational and rotational component of the slippage vector $\mathbf{v}_1$, respectively.

Building a Feature Graph and Bases: The next step of our algorithm is to build a graph of the detected line segments. The goal of this step is to connect "interesting" combinations of feature lines that are spatially close. For this, we connect every line segment to its k-nearest neighbors (We use $k = 100$). However, we limit the number of connections for segments representing the same line. For every connection, we check whether we have already connected to a line that describes the same circular arc and allow only one such connection per arc. As a result, we connect several lines of different type, even when they are farer away, while line segments describing the same feature lines are only connected locally to form a connected component.

Next, we perform a coarse segmentation of the line segments into longer pieces in order to identify longer and thus more relevant feature lines. For this, we walk along the graph of lines and group lines that lie on the same circular arc. We call these groups *line cluster*, representing a longer piece of a line feature of constant curvature. From these line clusters, we now form *bases*, which represent local coordinate frames

that will be used as matching candidates in later steps of the algorithm. Again, we walk along the graph and find all edges that connect between two line clusters for which their union is sufficiently non-slippable and store the pair of feature lines as basis. The same combination of two clusters is used only once (not once per individual line segment). However, we store two bases for each pair of clusters, with first and second coordinate axes exchanged. This allows us to handle mirroring. In order to reduce the number of spurious bases, we require that the two line clusters consist of a minimum number of line segments (typically, at least 4-8).

5. Line-Feature Matching

Having obtained a set of feature lines, our symmetry detection algorithm tries to find local constellations of these feature lines in different parts of the model that are similar. We use two steps: First, we identify a set of base matching candidates to estimate initial transformations. Second, we examine the line features in the neighborhood in order to verify the match. These two steps are iterated in a RANSAC like matching loop in order to find the best matches (this is the inner loop in Figure 2).

Base Matching: In order to find matching bases, we pick a random base out of the previously extracted set. Please note that large symmetries with many instances are automatically preferred in the random pick as they contain more bases. The base consists of two line clusters with different direction of slippability, therefore defining a local coordinate frame. Next, we find all other bases that could be potential matches, regarding only local information about the bases. For this, we build a simple feature descriptor: For each base, we store the curvature of the two lines involved and the average mean curvature of the surface points of the underlying geometry. We use a matching threshold for these values of $0.1/\varepsilon_{feat}$. In case both lines are straight lines with no curvature, we also include the angle at which the two lines meet in order make the criterion more discriminative. For each of these candidates with matching descriptors, we now compare the lines in its neighborhood by an alignment algorithm. Before going into details of the procedure, we first discuss the alignment algorithm:

Iterative Closest Lines (ICL): Our iterative closest line (ICL) algorithm is a straightforward generalization of the point-to-plane ICP algorithm [RL01] to aligning line segments. The idea of sampling well constrained regions is also similar in spirit to the adaptive ICP technique in [GIRL03]. We assume that we are given a point-sampled representation of curved lines: We have a set $\{\mathbf{x}_i\}$ of sample points on the lines, each equipped with a unit tangent $\mathbf{t}(\mathbf{x}_i)$, a unit normal $\mathbf{n}(\mathbf{x}_i)$ that describes the normal of the surface the line lives in. By construction, these two are orthogonal. From this, we also compute the normal $\mathbf{b}(\mathbf{x}_i) = \mathbf{t}(\mathbf{x}_i) \times \mathbf{n}(\mathbf{x}_i)$ to the feature curve in its tangent plane. Our goal is now to align two different sets of curved lines $\{\mathbf{x}_i^{(0)}\}, i = 1..n$ and $\{\mathbf{x}_j^{(1)}\}, j = 1..m$.

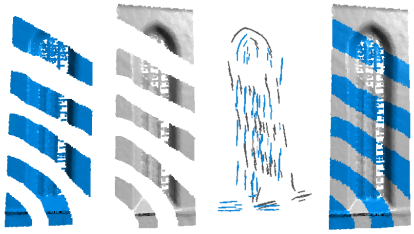

Figure 4: *ICL example: Aligning two disjoint subsets of a 3D scan. Maximum alignment error: 0.6% of the maximum bounding box side length. Third column: initial pose.*

Similar to the ICP algorithm, we first compute the closest point from the set $\{\mathbf{x}_j^{(1)}\}$ to each point from $\{\mathbf{x}_i^{(0)}\}$. We denote the index of the closest neighbor by $N(i)$. Given the correspondences, we minimize the following energy function:

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{n} \omega\left(dist(\mathbf{x}_i^{(0)}, \mathbf{x}_{N(i)}^{(1)})\right) \cdot \left[dist(\mathbf{x}_i^{(0)}, \mathbf{x}_{N(i)}^{(1)})\right]^2 \quad (1)$$

with

$$dist(\mathbf{x},\mathbf{y}) = \|[\mathbf{n}(\mathbf{x}) \,|\, \mathbf{b}(\mathbf{x}) \,|\, \mathbf{0}]\,(\mathbf{x} - (\mathbf{R}\mathbf{y} + \mathbf{t}))\| \quad (2)$$

where ω is a one-dimensional Gaussian window function, $\mathbf{R}$ is an unknown orthogonal matrix and $\mathbf{t}$ an unknown translation. The first matrix in Equation 2 is a projection matrix that removes the tangential component from the distance vector connecting two line segments, thus only measuring the point-to-line distance. For the optimization, we use the standard approach of locally linearizing the manifold of orthogonal matrices and solving iteratively [MGPG04].

RANSAC Search: As the set of feature line segments is much smaller than the number of original surface points, the ICL-alignment technique described above is significantly faster than a full alignment of the geometry. Therefore, we can afford to perform a large number of alignment tests in order to find matching candidates. We use this to perform a RANSAC-like randomized matching algorithm: We perform a number of iterations, searching for potential model symmetries. The output of each iteration is a list of candidate symmetries, which we score. In the end (after typically 100 trials), we keep only the best symmetries found.

Each iteration starts with computing a random base B_0, and a set of potentially matching bases $B_1 \ldots B_n$ for which the descriptors match. Next, we compute a pairwise matching score comparing the neighborhood of B_0 to all other bases B_i separately. Matching the neighborhood lines of two bases proceeds in several steps: First, we align only the lines that form the bases B_0 and B_i themselves, which can be computed analytically. At this point, the feature line representation has the advantage that matching pairs of lines already gives a stable initial guess, unlike feature point matching

[HFG*06, BBW*08]. Then ICL is used to refine the match. After aligning the lines of the bases B_0 and B_i, we walk on the graph of adjacent line segments in a region growing algorithm. We start by putting all line segments adjacent to those in B_0 on a priority queue sorted by distance. We then subsequently retrieve the next closest line segment from the queue and compute the weights of the point-to-line distances (Equation (1)) for the transformed lines. If the resulting weight ω (which now serves as *matching score* for this segment) is smaller than a fixed threshold (typically: 0.02), we discard the segment. Otherwise, we add the weight to the overall matching score of B_i and put its neighbors on the queue. The algorithm stops if the queue is empty or a maximum distance is reached (typically: 10% of the model size), to limit the costs.

After region growing, we perform a second ICL alignment step with all lines that were close to other lines (i.e., had high enough matching scores), refining the initial alignment. Next, the growing algorithm is performed again and might find further lines that became matches after the refined alignment. With this new matching set, a final alignment is computed and stored along with the match. If the overall matching score of B_i is too small in the end (< 50), the whole base-pair (B_0, B_i) is dismissed. Otherwise, we test if the base B_i is close to a base previously matched to B_0 (by computing the maximum point-to-line distance of the corresponding line segments). Here, close means in the range of sample spacing - we typically employ a threshold of $0.5\varepsilon_{feat}$. If such double matches are detected, we pick from the set of all colliding matches the one that achieved the largest matching score. This step avoids "ghosting artifacts" when matching objects with several parallel lines (for example window frames). Finally, if all these tests have been passed, we included the match $\{B_0, B_i\}$ into the set of found matches $M(B_0)$.

The previously described algorithm finds in each run a set of matches of bases along with rigid transformations that describe potential symmetries in the object (pending geometric verification on the full point set, described in the next section). The quality of the matches differs strongly, depending on which base B_0 had been chosen as starting base. Therefore, we execute the algorithm several times (typically $100\times$) and output only the best match found to the next stage, geometric validation, which is more costly. The score this decision is based on is computed as:

$$score = matching\_score_{av} \cdot \#instances^2 \qquad (3)$$

$matching\_score_{av}$ refers to the previously determined sum of weights of the matched lines, averaged over all valid instances out of $\{B_1 ... B_n\}$. It grows if more matching lines are found per instance. The number of instances is the number of valid base matches found. By squaring, the number of instances gets a stronger weight; we prefer slightly smaller instances if this allows us to find many more of them. The rationale behind this is that finding one more instance makes a symmetry more plausible than just adding one more line segment.

6. Geometric Validation

So far, we have done all computations on the line features of the surface $\mathcal{S}$ only. This usually gives a good indication of actually corresponding geometry but we now need to verify this on the full geometry, i.e., the original point set. At this stage, we will also form concrete, disjoint pieces of geometry that form the instances of a symmetry. As input to this stage, we are given a set of bases $\{B_0, ..., B_k\}$ and their associated (ICL-refined) transformations $\{T_1, ..., T_k\}$ between B_0 and $\{B_1, ..., B_k\}$. The output of the algorithm is a single point cloud, which we call *urshape*, that fits parts of the original geometry $\mathcal{S}$ when being transformed by any of the transformations $\{\mathbf{I}, T_1, ..., T_k\}$. The transformed instances are disjoint and the urshape is maximal, i.e., cannot be extended without exceeding a given error threshold.

Basic region growing: We perform region growing, starting at the bases, and collect all points that match in other instances. In order to start growing, we first choose one point of the basis B_0 as a *reference point*. Next, we transform the reference point into all other instances. In each instance, including B_0, we compute the sample point from the sample manifold $\mathcal{S}$ that is closest to the respective transformed reference point. We will then use these points as starting points for region growing and stop if points are already occupied by another instance or the transformed geometry does not match. The region growing will proceed by Euclidian distance to the starting points, which yields nice, Voronoi-type boundaries. We initiate growing by inserting the starting points into a priority queue. The queue is sorted by Euclidean distance to the corresponding reference points in each instance. Afterwards, we iteratively extract points of minimum distance from the queue, transform it into all instances, and test for geometry mismatch and collisions with different instances. Only if neither is the case, growing will continue.

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 instable 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 $(T)_i^{-1}$ 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.

Grid based growing: The basic region growing algorithm with its test of every point is too expensive for huge models. We improve the performance by 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.

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 (typically: less than 6), 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 (up to 30% in terrestrial scanner datasets), allowing for matching partial data more robustly.

Refinement: A drawback of the algorithm as presented so far is that it can only detect a class of instances that all contain the same basis (in the sense of a pair of lines with the same curvature, placed roughly at the same spatial location). In order to improve the recognition performance for difficult data sets, we now include adjacent bases into the search (this is the "second pass" in Figure 2): After geometric validation, we know the concrete area covered by the instances. Therefore, we can check which other bases are also covered by the geometry and try to match them as well. We compute all bases covered by an instance and project them back into the urshape. In the urshape, we cluster similar bases, i.e., consisting of approximately the same lines. For each cluster, we maintain a counter to vote for bases that are contained in many symmetric instances. Having a weighted list of other bases contained in the same instance, we now execute the line matching algorithm of Section 5 again, starting at these alternative basis. Instead of region growing, we directly fix the area to be checked by the line segments that fall onto the urshape when being back-projected. In order to limit the computational costs, we do not check all alternative bases (which can be thousands), but rather sample a small number of bases (typically 5) randomly, with probability proportional to the number of votes for each basis. After this step, we execute region growing again, now including all newly found instances.

We can improve the recognition rates further using a prediction heuristic, inspired by [PMW*08]: We consider the relative transformations between instances that are spatially close and recursively check whether we can find a matching line constellation if we apply this transformation again to an instance at the boundary of the detected area (a candidate transformation at a boundary will transform a base such that it does not come close to an already known base).

In order to evaluate the utility of the different passes, we have looked at the percentage of instances detected in each step, relative to the overall number found. For the "old townhouse" data set, for example, the basic algorithm itself already detects all of the finally detected instances in about 75% of the symmetric parts. In the remaining cases, looking at nearby bases accounted for 40%, in one case 80% of the matches. Prediction detected 25%-40% in the cases where basic matching was not successful. For other datasets, we obtain comparable results; prediction is of course most useful in architectural data sets with regular pattern. However, the technique is not limited to fixed grids in transformation space but also works for partially regular data. Even without any successful prediction, we are able to find most detectable instances in the majority of the cases.

7. Candidate Loop and Outer Loop

So far our algorithm is able to detect a single urshape and a set of transformations that transform this urshape into places where symmetric geometry exists, giving a single symmetry pattern. We now employ an outer loop to iteratively detect many such patterns. We execute the whole algorithm described so far multiple times, tagging all points and nearby bases and line segments as "visited" in each validation (growing) step.

Candidate loop: We can make this basic strategy more reliable by again employing a RANSAC-like randomized sampling approach: Instead of removing and thus finalizing symmetric geometry after finding one new set of matches, we first compute a larger number of matches (typically 5 matches), including geometric validation (the refinement step described in the preceding paragraph is actually done *after* the candidate loop, on its result, to safe some computation time; see Figure 2). From the candidate set, we pick only the best match. The score for this decision is computed as number of (voxel-quantized) sample points in the urshape multiplied by the number of instances squared (similar to Equation (3), again preferring many instances over few instances with many points. The outer RANSAC loop stops when a fixed number of outer loop iterations has passed. We do not accept symmetries covering fewer than a minimum number of data points (less than 1,000) to avoid spurious matches. This prevents outputting "garbage" solutions after all detectable symmetries have been found. An interesting side effect of this strategy is that the algorithm will output the "most important" symmetries first, which are symmetries with many instances covering a lot of area.

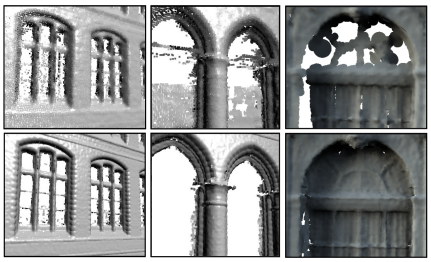

Figure 5: *Reconstruction examples for the data sets old town hall and Zwinger (rightmost).*

8. Applications

As an application of our symmetry detection algorithm, we consider an automatic reconstruction-by-symmetry algorithm, in the spirit of [TW05, PMG*05, GSH*07, PMW*08]. Many real-world range scans suffer from strongly irregular sampling, noise and acquisition holes. We will automatically detect similar parts and compute a high density average to improve the sampling quality in regions where such redundant information is available.

Our reconstruction technique first computes an improved urshape and then transforms the improved version back into the instances, replacing the distorted geometry. We start the urshape improvement by copying all points of all instances into the urshape, 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-occurance test, we use a small sphere around the sample point, according to the scanner sample spacing, and check whether the instance provides such a supporting point within that radius. The remaining non-outlier points are then smoothed using a quadratic MLS approximation.

9. Implementation and Results

We have implemented the proposed symmetry detection and reconstruction system in C++. Timings have been obtained on a standard PC with Core2 Duo at 2.4Ghz, using a single threaded implementation. In order to visualize the results, we assign a unique color to each symmetry. If two instances of the same type share a border, we indicate this by drawing a black/white colored line in between. In the following, we discuss the results for several test data sets.

Synthetic example: As a synthetic example for completely irregular patterns of symmetry, we have engraved the Eurographics logo into a flat plane, rotated and shifted randomly. For some of the instances, we have cut out holes of different size. Our algorithm is able to detect all of the logos automatically (Figure 8); it discovers the reflective symmetry of the logo, leaving non-symmetric parts in the letter G out, and detects all instances of these in the scene. If we add Gaussian noise with a standard deviation of 5% of the maximum heightfield height, some of the instances with big holes are not retrieved anymore but we still obtain good results for most of the symmetric parts.

Real world scanner data: We also apply the proposed algorithm to raw 3D scanner data. Our first example is a scan of a museum (Figure 6, left). We are able to recognize most of the apparent symmetries, including most of the possible instances (such as the windows in the front). In particular, we detect instances as belonging to the same class that are not part of a simple regular pattern (see for example the jutty in the middle of the building). A remaining limitation of our approach is the fixed allocation of geometry to a single instance. Therefore, some choices of forming instance preclude detecting some other symmetries (for example: the small window above the jutty, which is not combined with a larger window below, unlike most other of these small windows). This effect is responsible for most of the missed out symmetries in this example. Please note that we do not obtain any false positives. This shows that the multi-stage filtering strategy of our algorithm is successful in that respect.

Our second test case is a scan of the old town hall in Hannover (Figure 7). This is our largest example. In order to bound the computation time, we use a fixed number of 25 outer loop RANSAC iterations, which yields 19 successfully detected (i.e., large enough) instance sets. In this case, we detect may symmetries with a large number of instances. However, we also obtain some symmetries covering a large area with a small number of instances (for example the facade to the left). These regions mostly consist of small windows with few horizontal lines which makes a detection of the small scale instancing structure harder, so that the initial set (before refinement) is not likely to contain many instances. Therefore, the large symmetry often wins the outer loop RANSAC decision by its area. Again, this problem could be resolved by relaxing the restriction of simple, non nested and non-overlapping instances, which we leave for future work.

The last example is the Zwinger data set from [BBW*08]. In comparison to Berner et al.'s approach, we obtain significantly better recognition results (detecting all windows instead of only two instances).

To examine the limits of our algorithm, we have applied it to the Happy Buddha data set. This yields only two small reflective symmetries (Figure 3). Line detection and matching works in principle for such data sets, but the main problem in this case is that the data set does not contain larger rigidly symmetric parts.

Reconstruction: We apply our reconstruction approach to the front of the old town hall and to the Zwinger data set (Figure 5). With our symmetry reconstruction technique we obtain a significantly improved quality. Surprisingly, the quality is better than one would expect from just averaging a small number of instances (just leading to square-root error

Figure 6: *Detected symmetries in real world 3D scans – left: Museum (data set courtesy of the Institute for Cartography and Geoinformatics Hannover). right: Dresden Zwinger (data set courtesy of M. Wacker).*

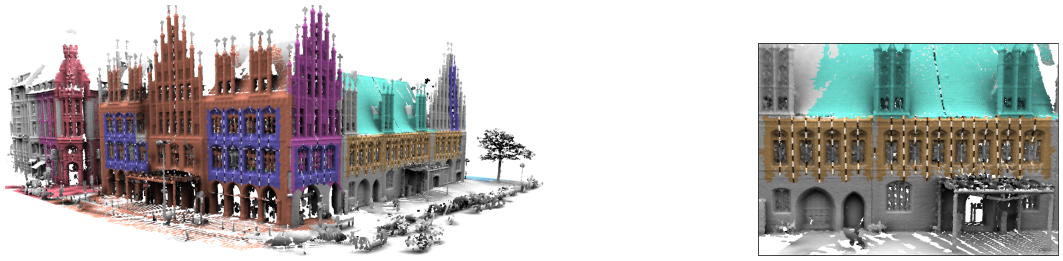

Figure 7: *Old Town Hall Hannover (courtesy of the Institute for Cartography and Geoinformatics Hannover).*

decay). The reason for this is that we do not only average out noise, but also increase the sampling density; in these examples, an insufficient sampling density lead to more geometric uncertainty than the noise in the distance measurements. In addition, we can also fill holes such as the window frames, which are typically acquired from at one side only.

Performance: Statistics and timings for the example scenes are shown in Table 1. Our largest data set, the old town hall, took about 50 minutes to process, with a little less than half of the time spend in feature detection. In comparison to [PMW*08], we are able to handle scenes that are substantially larger than the data sets examined in their paper. In terms of running time, we have comparable requirements, taking the size of the model into account.

Parameters: Our algorithm depends on a number of parameters. Most of these parameters are constants for all data sets, using the "typical" values given in the text above. There are a few parameters that have to be set manually for each data set. The basic parameter is a resolution parameter ε_{res}, from which most others are derived: This parameter has to be set to the sample spacing or the noise level of the input data (whatever is larger). Currently, we estimate this manually and use a constant for the whole scene. Given this quantity, we set ε_{feat} as well as the voxel size for region growing to $5\varepsilon_{res}$ and set the standard deviation of the weighting windows in ICL to ε_{res}. For very noisy data sets, enlarging this value might slightly improve the results; for the noisy logo test scene (only there), we use $1.5\varepsilon_{res}$. We also use ε_{res} as threshold for the maximum distance of lines in clustering the line segments. There is one more spatial parameter proportional to ε_{res}: the distance threshold in region growing, which determines the allowed geometric variation in instances. A value of $2\varepsilon_{res}$ usually gives good results, but there is a delicate trade-off between too small instances and missing small features. We think that improving upon this might require a global optimization technique for laying out the instance shape, for example using a graph cut in the urshape domain, which we have to leave for future work. A last set of parameters is the number of RANSAC loops. We always use 100 iterations of the inner loop and 5 for the candidate loop; more can only improve the results, at higher costs. The number of outer loops limits the number of detected instances. We use an exhaustive number of iterations (i.e., the algorithm stops finding new instances before terminating) in all examples except for the largest one, as discussed above.

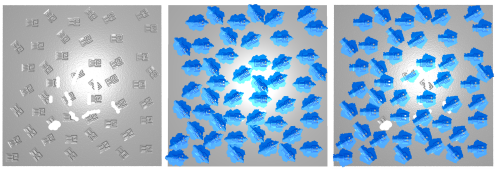

Figure 8: *Synthetic data set – left: noisy input data, middle: recognition results for noise free data, right: results for noisy data.*

10. Conclusions and Future Work

In this paper, we have presented a new algorithm for symmetry detection. The main idea is to look for symmetric constellations of feature lines on 3D surfaces in order to find

	# Points	Feature Detection	Clustering + Graph	Symmetry Detection
Plate	880.673	25s	1.1s	1m 59s
Plate (noise)	880.673	1m 12s	3.3s	3m 14s
Old townhall	7.735.576	23m 12s	25.8s	32m 40s
Museum	2.200.652	4m 44s	5.2s	13m 36s
Zwinger	278.512	32s	2.3s	3m 25s

Table 1: *Statistics of the datasets used in this paper.*

similar parts. In comparison to previous transformation voting algorithms, we avoid the problem of cluttering the transformation space and therefore get a good recognition performance without additional assumptions on the structure of the symmetries. In comparison to previous attempts of using feature points for symmetry detection, feature lines yield a significant improvement in recognition results. As a side effect, the reduction to feature lines reduces the amount of data to be examined substantially, allowing for handling substantially larger models than previous algorithms. A limitation of our algorithm is that the reconstruction only works if instances are all exactly similar. In future work, we would like to build morphable statistical models of symmetric parts to better represent subtle variations beyond the accuracy of the region growing algorithm. In addition, we would also like to examine more general strategies in matching graphs of surface features, beyond rigid mappings.

Acknowledgments

The authors would like to thank Benjamin Jogsch for his help and the anonymous reviewers for their valuable comments. This work has been supported by DFG grant *Perceptual Graphics* and the Cluster of Excellence for Multi-Modal Computing and Interaction.

References

[BBW*08] BERNER A., BOKELOH M., WAND M., SCHILLING A., SEIDEL H.-P.: A graph-based approach to symmetry detection. In *Proc. Symp. Point-Based Graphics 2008* (2008).

[BM92] BESL P. J., MCKAY N.: A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (1992).

[CM92] CHEN Y., MEDIONI G.: Object modelling by registration of multiple range images. *Image Vision Comput. 10*, 3 (1992), 145–155.

[FB81] FISCHLER M. A., BOLLES R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Comm. ACM 24*, 6 (1981).

[GCO06] GAL R., COHEN-OR D.: Salient geometric features for partial shape matching and similarity. *ACM Trans. Graph. 25*, 1 (2006), 130–150.

[GG04] GELFAND N., GUIBAS L. J.: Shape segmentation using local slippage analysis. In *Proc. Symp. Geometry processing* (2004), pp. 214–223.

[GIRL03] GELFAND N., IKEMOTO L., RUSINKIEWICZ S., LEVOY M.: Geometrically stable sampling for the icp algorithm. In *Proc. Int. Conf. 3D Digital Imaging and Modeling* (2003).

[GSH*07] GAL R., SHAMIR A., HASSNER T., PAULY M., COHEN-OR D.: Surface reconstruction using local shape priors. In *Proc. Symp. Geometry Processing* (2007).

[GWM01] GUMHOLD S., WANG X., MACLEOD R.: Feature extraction from point clouds. In *Proc. Meshing Roundtable* (2001).

[HFG*06] HUANG Q.-X., FLÖRY S., GELFAND N., HOFER M., POTTMANN H.: Reassembling fractured objects by geometric matching. *ACM Trans. Graphics 25*, 3 (2006), 569–578.

[HPW05] HILDEBRANDT K., POLTHIER K., WARDETZKY M.: Smooth feature lines on surface meshes. In *Proc. Symp. Geometry Processing* (2005).

[KCD*03] KAZHDAN M., CHAZELLE B., DOBKIN D., FUNKHOUSER T., RUSINKIEWICZ S.: A reflective symmetry descriptor for 3d models. *Algorithmica 38*, 1 (2003), 201–225.

[LE06] LOY G., EKLUNDH J.: Detecting symmetry and symmetric constellations of features. In *Proc. Europ. Conf. Computer Vision* (2006), pp. 508–521.

[LW88] LAMDAN Y., WOLFSON H. J.: Geometric hashing: A general and efficient model-based recognition scheme. In *Proc. Int. Conf. Computer Vision* (1988).

[MGP06] MITRA N. J., GUIBAS L. J., PAULY M.: Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph. 25*, 3 (2006), 560–568.

[MGP07] MITRA N. J., GUIBAS L., PAULY M.: Symmetrization. In *ACM Transactions on Graphics* (2007), vol. 26.

[MGPG04] MITRA N. J., GELFAND N., POTTMANN H., GUIBAS L.: Registration of point cloud data from a geometric optimization perspective. In *Symp. Geometry Processing* (2004).

[MSHS06] MARTINET A., SOLER C., HOLZSCHUCH N., SILLION F.: Accurate detection of symmetries in 3d shapes. *ACM Trans. on Graphics 25*, 2 (2006), 439 – 464.

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Ridge-valley lines on meshes via implicit surface fitting. In *Proc. Siggraph* (2004), pp. 609–612.

[PKG03] PAULY M., KEISER R., GROSS M.: Multi-scale feature extraction on point-sampled models. In *Proc. Eurographics* (2003).

[PMG*05] PAULY M., MITRA N., GIESEN J., GROSS M., GUIBAS L. J.: Example-based 3d scan completion. In *Proc. Symp. Geometry Processing* (2005).

[PMW*08] PAULY M., MITRA N. J., WALLNER J., POTTMANN H., GUIBAS L.: Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics 27*, 3 (2008).

[PSG*06] PODOLAK J., SHILANE P., GOLOVINSKIY A., RUSINKIEWICZ S., FUNKHOUSER T.: A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proc. SIGGRAPH) 25*, 3 (2006).

[RL01] RUSINKIEWICZ S., LEVOY M.: Efficient variants of the ICP algorithm. In *Proc. 3rd Intl. Conf. 3D Digital Imaging and Modeling* (2001), pp. 145–152.

[SKS06] SIMARI P., KALOGERAKIS E., SINGH K.: Folding meshes: hierarchical mesh segmentation based on planar symmetry. In *Proc. Symp. Geometry Processing* (2006), pp. 111–119.

[SWWK08] 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).

[TW05] THRUN S., WEGBREIT B.: Shape from symmetry. In *Proc. Int. Conf. Computer Vision* (2005).