Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Bachelor thesis

Extracting Point Features for Symmetry Detection

submitted by

Daniel Mewes

submitted

November 8th 2010

Supervisor

Dr. Michael Wand

Advisor

Alexander Berner

Reviewers

Prof. Dr. Hans-Peter Seidel
Dr. Michael Wand

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement under Oath

I confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,..... .....

(Datum / Date) (Unterschrift / Signature)

I thank Alexander Berner and Michael Wand for their helpful comments and advise.

I would further like to thank everyone at the Max-Planck-Institut für Informatik Saarbrücken for maintaining an extraordinary friendly and inspiring workplace atmosphere.

Table of Contents

1 Introduction

The increasing demand for high-quality 3D models for use in computer games, navigation, augmented reality applications etc., makes data acquisition through real-world digitalization increasingly useful. Robust symmetry detection provides a powerful tool to support processing, editing and storage of such data. Symmetry detection poses the problem of automatically detecting repeating structures in a scene. Specifically, such information can be used for efficient data compression, simultaneous 3D model editing (modifications to one instance of a symmetry are automatically transferred to other instances), intelligent data reconstruction and/or automated scene modeling.

While symmetry detection is computationally expensive – especially if imperfections in the input data have to be accounted for – the extraction of features allows to significantly reduce the amount of data that has to be considered. This is usually done by first selecting candidates for symmetries based on the set of discrete features, and then validating those candidates against the complete scene data. Unfortunately, this approach has the problem that the set of symmetries detected in the feature data imposes an upper bound on which symmetries can be detected in the underlying scene.

We introduce the notion of a set of symmetry features to describe feature points which are consistent across all instances of some symmetry in the scene. A set of symmetry features ensures that symmetries in the underlying point cloud also impose a corresponding symmetry structure in the set of feature points and therefore enables subsequent point-feature-based symmetry detection to detect exhaustive sets of symmetries.

It turns out that the extraction of symmetry feature points requires knowledge about the global structure of the scene. We propose a framework that interleaves preliminary feature point extraction and partial symmetry detection in an iterative way. Additionally we introduce the idea of small-scale sub-symmetries and propose an algorithm to detect such. A sub-symmetry is a fixed-size constellation of corresponding feature points. By overlaying the information we get from a large number of sub-symmetries, we are able to retrieve robust structural information that we employ within our framework to improve the extracted feature sets.

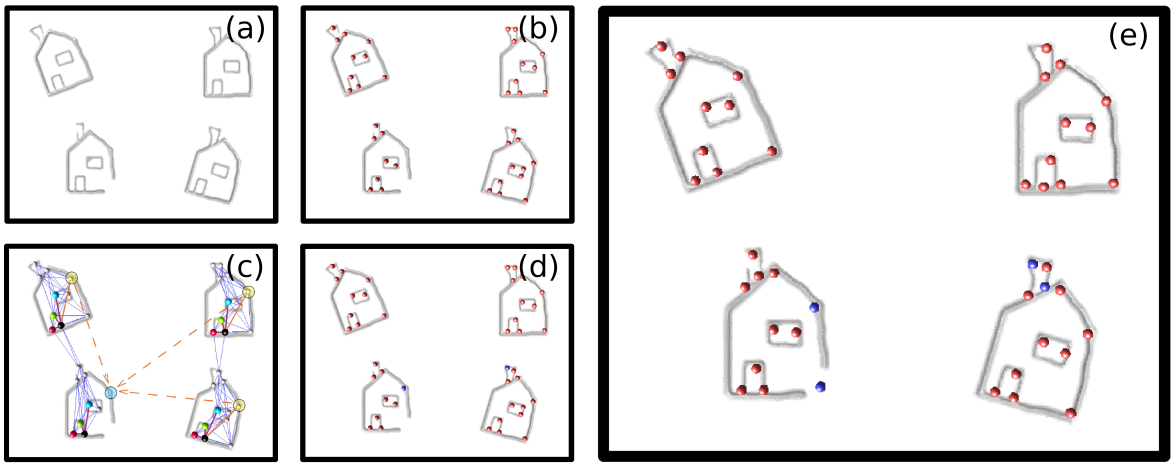

Figure 1: Example for intermediate results of our pipeline

Our approach to symmetry feature extraction can be summarized as follows:

Given a point sampled 3D scene as our input data (figure 1a), we first calculating local per-point feature scores. For this we employ existing algorithms, namely local slippage analysis ([Gelfand et al. 2004]) and Gaussian curvature estimation. Based on the local scores, we extract an initial set of feature points (figure 1b). Using these features, we apply global sub-symmetry detection. By this, we get information about the scene's structure, which we use to generate stochastic guesses about additional features. Figure 1c shows an example of how such guesses can be derived from sub-symmetry information: The black, red, yellow and blue points mark instances of a detected sub-symmetry. For three of the four instances, the feature point under the yellow circle exists in the neighborhood. A guess about a corresponding feature point in the fourth instance at the position of the blue circle is derived from that information.

Having extended our knowledge about potential features, we incorporate all guesses into the previously calculated per-point scores and select a refined set of feature points (figure 1d). We iterate the steps of symmetry detection and refined feature selection for a number of times, until the final set of feature points is achieved (figure 1e).

Applying an implementation of our framework to scanned real-world data, we were able to significantly improve the set of extracted feature points over the feature sets selected from local-criteria only. Figure 2 shows an example of feature points extracted for a scanned real-world 3D data set. Furthermore, we were able to utilize sub-symmetry detection to extract feature-to-feature correspondence classes. While those lack a whole-scene-based validation in our implementation, they can provide a starting point for subsequent full-fledged symmetry detection.

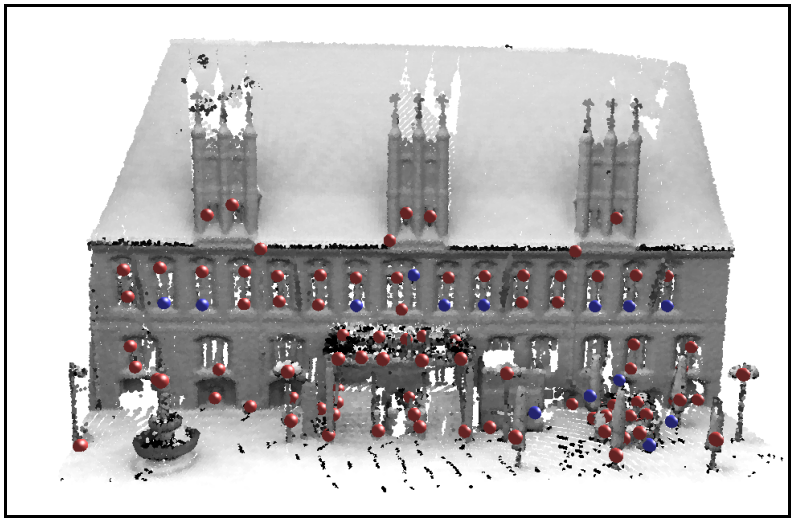

Figure 2: Blue feature points were selected with the help of global symmetry information. We were able to recover all eight missing feature points in the upper row of windows. (old Hannover town hall data set)

The structure of this thesis is as follows:

In section 2 we give a short overview on related work from the areas of symmetry detection and feature extraction. Section 3 contains an explanation of the exact problem we are trying to solve within this thesis. Our solution to this problem is presented in section 4, where we introduce our framework and provide detailed documentation of all of its parts. Implementation details which are not required for an understanding of our framework are provided in separate boxes. In section 5, we present results of the framework using different data sets and parameters and discuss relevant observations. We give a brief conclusion on the results of this thesis and provide an outlook on future research in section 6.

2 Related Work

An early approach for detection of approximate rigid symmetries is described in [Alt et al. 1988]. They introduce the notion of "approximate congruence" and give a formal definition of such. The runtime complexity of their algorithm is shown to be in $O(n^8)$ for the case of two-dimensional point sets of no more than n points. For translational symmetries the complexity of their algorithm is still $O(n^6)$. While their paper provides a lot of useful formalisms and extensive proofs, the runtime complexity of the proposed algorithms essentially rule out the application to large data sets.

A more practical approach is proposed by [Mitra et al. 2006]. They reduce the problem of approximate symmetry detection for three dimensional point sets to finding clusters in a seven-dimensional transformation space. The basic idea is known as "transformation voting" and in the case of [Mitra et al. 2006] works as follows (compare figure 3):

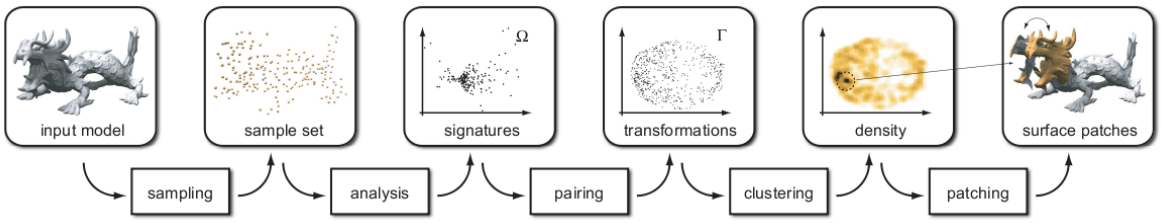

Figure 3: Symmetry extraction pipeline from [Mitra et al. 2006]

Given a 3D point set, first perform sub-sampling for improved practical performance. For the subset, generate local per-point normal and principal curvature data (assuming the point set is a manifold). Then pairs of points are considered. For each such pair, a vector in $\mathbb{R}^7$ is generated, which describes a scaled rigid transformation based on a scaling factor derived from the ratio between both points' principal curvatures, three rotational components which are derived based on both points' normal and principal curvature directions (those form a local coordinate frame which is invariant under rigid transformation and scaling), and three translational coordinates. All generated pairs are placed in a unified seven-dimensional transformation space. Clusters of point-pairs in this space give candidates for rigid symmetries, which are then brought over to the original point set.

[Pauly et al. 2008] employ a different kind of transformation-based voting to detect one- and two-dimensional grid structures in 3D shapes. A one-dimensional grid structure over a shape $\Omega \subset \mathbb{R}^3$ can be characterized by a transformation $f : \mathbb{R}^3 \to \mathbb{R}^3$ and an initial instance

$S \subset \Omega$, such that $f^i(S) \subset \Omega$ with $i \in \mathbb{N}$ at least for all $i < n, n \in \mathbb{N}$. In other words: the transformation f can be repeatedly applied on the instance S without leaving the shape. Two-dimensional grids have two such transformations $f_1, f_2 : \mathbb{R}^3 \to \mathbb{R}^3$ which can be applied on S in any order while still giving subsets of Ω. In order to identify such grid transformations in a scene, [Pauly et al. 2008] detect "line structures" in transformation space. More specifically, they detect a grid transformation $f : \mathbb{R}^3 \to \mathbb{R}^3$ represented by its scaling $s \in \mathbb{R}$, rotational $R \in \mathbb{R}^3$ (one angle per canonic axis) and translational $t \in \mathbb{R}^3$ components by looking for incidental transformations $f' = f^i, i \in \mathbb{N}$. These transformations are exactly those which have a scaling factor of $i\,s$, a rotational component of $i\,R$ and a translational component of $i\,t$.

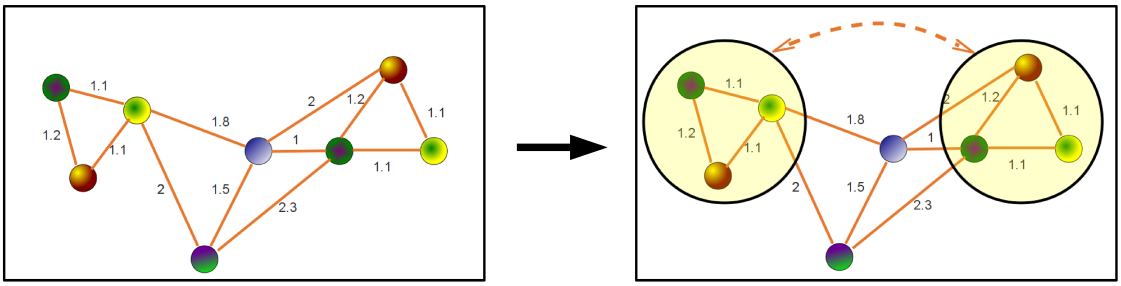

Figure 4: Isometric subgraph detection

Approaches based on transformation voting have the general disadvantage of neglecting the spatial relation between points in the scene. Point-pairs from completely unrelated locations can still end up at the exact same position in transformation space. [Berner et al. 2008] take a different approach to symmetry detection. They employ feature points and explicitly utilize the locality of feature points participating in any single instance of some symmetry. Their overall idea is as follows: First extract a set of feature points (here: slippage features, see below) together with a local feature point descriptor. Then build a graph over the feature points by connecting spatially nearby features with an edge and annotate each edge by its length. Figure 4 shows a schematic example for such graph. Symmetry candidates are identified by looking for repeating subgraphs, for which each instance matches with respect to the feature descriptors and the edge lengths. To make the detection of such isometric subgraphs computationally feasible, a RANSAC approach is taken. Feature-based symmetry candidates are then transferred to the whole point set and verified. Our approach to symmetry detection (see chapter 4.8.4) is inspired by [Berner et al. 2008] in the sense that we also consider the metric relation between feature points to detect symmetry candidates.

The problem of point feature extraction is tackled for example by [Lowe 1999]. Lowe extracts feature key points from 2D image data by looking for points which are local extrema consistently across variants of the input image filtered with Gaussian kernels of different scale. The key-points are annotated with a local descriptor to form "SIFT" features.

[Bokeloh et al. 2008] perform local slippage analysis on 3D point sets and select points of low slippability as feature points. Local slippage analysis as proposed by [Gelfand et al. 2004] provides a metric on how well the alignment of a small local patch to itself is constrained. We utilize slippage analysis within our framework to get a starting point for feature extraction. In chapter 4.8.1 we will further describe the idea of slippage analysis.

[Yilmaz et al. 2009] establish a global stochastic model which incorporates local texture data and location information, in order to robustly extract facial features from image data. While the idea of globally optimized feature detection is comparable to what we want to achieve, their method requires specific training for the single kind of objects that features are to be extracted for.

A lot of work also exists about the extraction of non-point features from point sets, for instance [Gumhold et al. 2001] or [Bokeloh et al. 2009] for line feature extraction. For simplicity reasons, we only consider point features within this thesis however.

3 Problem Definition

Automatic symmetry detection is the problem of finding repeating structures in a scene.

One representation for 3D scenes is a finite point cloud $\Omega \subset \mathbb{R}^3$. This form of representation especially has the advantage of being easily obtainable for real-world objects by the use of 3D scanners. Also most other representations – like triangle meshes, implicit surfaces etc. – can be easily sampled as a point cloud. In practice, most scenes are two-dimensional manifolds essentially. Triangle meshes for example do not even support non-manifold scenes. The manifold property allows to define additional per-point properties such as normal directions and local curvature. In the following we assume the scene represented by Ω to be a 2D manifold.

A symmetry over Ω can be described as a set $S \subset \Omega$ together with a number of transformations $f_1, \ldots, f_n : S \to \Omega$. Automatic symmetry detection is about finding such subsets with corresponding transformations given the scene $\Omega \subset \mathbb{R}^3$.

For practical relevance, a symmetry should be "meaningful". In any scene Ω, there are $2^{|\Omega|}$ different subsets $S \subset \Omega$, each allowing for at least an exponential number of corresponding transformations. Additional criteria are required to make the given definition of a symmetry useful. As a first step we limit the set of potential transformations. Intuitively, one would for example expect that point neighborhoods are pertained by each f_i. More exactly: if $x, y \in \Omega$ are near to each other, $f_i(x)$ and $f_i(y)$ should be near to each other too. Analytically speaking, f_i should be a continuous mapping. Apart from this criteria for potential symmetry transformations, the set S should be of a certain minimum size to form a "meaningful" symmetry.

One subset of symmetry transformations are rigid transformations. A rigid transformation is a composition of translational, rotational and mirroring transformations. For a rigid transformation $f_i : S \to \Omega$, it follows that for every two points $x, y \in S$ it is $\|x - y\| = \|f_i(x) - f_i(y)\|$. A symmetry where $f_1, \ldots, f_n$ are rigid transformations, is called a rigid symmetry. In practice, the sampling Ω of the underlying scene as well as noise in the data usually make such symmetries impossible to find. More practical are such symmetries which have transformations that are approximately rigid, with $\|x - y\|$ and $\|f_i(x) - f_i(y)\|$ being equal up to some error. Whenever speaking of rigid symmetries in the following, we actually refer to approximate rigid symmetries.

Applications for symmetry detection range from simultaneous editing, over data compression, to scan data reconstruction. Symmetry information has also been used to perform automated modeling, for example in [Bokeloh et al. 2010].

Unfortunately, symmetry detection is a computationally expensive process. Even in the comparably simple case of rigid symmetries, a minimum of one pair of corresponding points is required to define a rigid transformation f_i. This is for points with additional normal and principal curvature information available, as this information is required to constrain the rotational and mirroring components of a rigid transformation. If no such information is available, a total of six points (three points corresponding to three others) is required. Given a transformation f_i and a pair of initial points $x, y \subset \Omega$ with $f_i(x) = y$, a corresponding set of points $S \subset \Omega$ such that $f_i(S) \subset \Omega$ can usually be derived by successive area growing, This works by adding neighbor points to S successively, each time checking that the criteria $f_i(S) \subset \Omega$ is still met. Even then, just checking all potential symmetry transformations has runtime complexity in $\Theta\left(|\Omega|^2\right)$ at least. Various techniques exist to minimize the cost of symmetry detection. Examples are:

- Randomized approaches, especially RANSAC ([Fischler et al. 1981])
- Operating on a downsampled subset of Ω (e.g. [Mitra et al. 2006], [Pauly et al. 2008])
- Feature-based approaches (e.g. [Berner et al. 2008], [Bokeloh et al. 2009])

These techniques can also be combined. For example [Bokeloh et al. 2009] use a RANSAC candidate loop over a feature set.

A set of features provides a characteristic subset of all available information about the scene (here: Ω). In order to deploy features in symmetry detection, the idea is to first detect symmetry candidates from the feature set. Then, only those candidates have to be verified against the complete data. While there are different classes of features, the conceptually simplest one are point features. A set of point features Ξ for the scene $\Omega \subset \mathbb{R}^3$ is a subset of $\mathbb{R}^3$. While not strictly necessary, we restrict Ξ to points in Ω for practical reasons (i.e. $\Xi \subset \Omega$).

To increase the descriptiveness of a feature point, a transformation invariant descriptor can be added to each feature point, incorporating information about the point's environment. [Bokeloh et al. 2008] for example employ a curvature-based descriptor. Often the word feature point is used to describe the combination of a point $\xi \subset \Xi$ together with its descriptor, while ξ alone is called a key-point to distinguish between both. Here, we will also call the point alone a feature point, as descriptor information is not used throughout this thesis.

3.1 Symmetry Features

Let $\Omega \subset \mathbb{R}^3$ be a finite set of points in 3D-space, which describes the scene we want to analyze.

Let there be subsets $S_1, \ldots, S_n \subset \Omega$ and corresponding mappings $f_1 : S_1 \to \Omega, \ldots, f_n : S_n \to \Omega$ which describe some symmetries in the scene.

We call a set of feature points $\Xi \subset \Omega$ a set of *symmetry features* with respect to these symmetries, whenever we have $x \in S_i \cap \Xi \Rightarrow f_i(x) \in \Xi$, which is all symmetry transformations $f_1, \ldots, f_n$ are closed in Ξ.

The definition of symmetry features ensures that in order to detect symmetries in Ω, it is possible to first search symmetries in a corresponding set of symmetry features and by doing so generating a superset of all the symmetries which exist in the actual scene .

3.2 Problems in the Extraction of Symmetry Features

Depending on the scene to be analyzed, it can be impossible to locally (i.e. by just considering properties of the point itself and some fixed-size neighborhood) decide if any given point of the scene is to be considered a symmetry feature. There are two classes of problems which may arise when trying to do so:

1. Noisy and/or erroneous data: It is desirable to be able to extract symmetry features in real-world scenes. In order to do this, the scene first has to be digitalized. This is often done using 3D laser scanners. Unfortunately, the scanning process incorporates various sources of errors. For example both reflective and very dark material impose a problem for laser scanners. These errors can lead to noisy or badly constrained data points. Such errors in point positions are not always recoverable by means of local post-filtering, as errors can be biased and/or indistinguishable from actual structure.

2. Structural problems: Sometimes the local structure of the scene itself makes it impossible to extract a complete set of symmetry features using local criteria. While there might be no local indication for a certain point to be a symmetry feature, the global symmetry structure of the scene can still make it desirable to selected such point. Figure 5 illustrates the problem: While for the "inner" scales, the red points qualify locally as significant feature points, there is no such indication for a corresponding feature point on the "outer" scales. To obtain a valid set of symmetry features, the detection of the points marked yellow would be necessary.

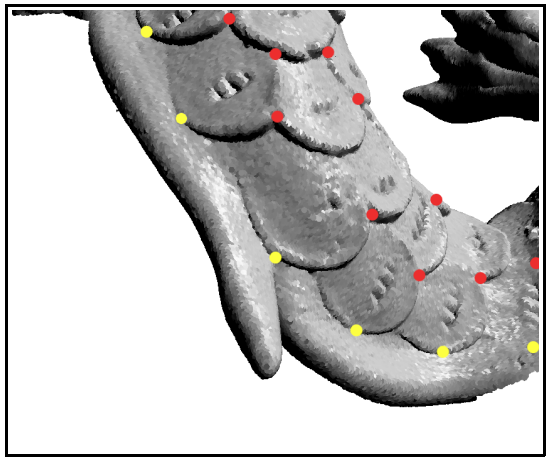

Figure 5: Structural problem example (Asian Dragon model from the Stanford 3D Scanning Repository)

We propose a framework to extract approximate sets of symmetry features while working around these two classes of problems.

As a third class of problems, there might be holes in the scene. Using our definition of symmetry features from chapter 3.1, which only speaks about points contained in the scene, this does not strictly impose a problem for extracting symmetry features. In practical applications it is often desirable to work around holes nonetheless. Our framework has no direct support for fixing this kind of error. However it can often be reduced to a problem of class one by filling holes using for example Poisson reconstruction as described in [Kazhdan et al. 2006].

3.3 Use Cases and Advantages

The designated way of use for our framework is the extraction of feature points for use in a subsequent symmetry detection step. As features are usually used to derive a superset of symmetries in the scene, it is important that the feature-based symmetries do not miss a valid symmetry candidate. While having a set features which is too large can usually be handled well by symmetry detection algorithms (it only leads to additional symmetry candidates, which are filtered out during validation), the superfluous features reduce the performance of symmetry detection. [Berner et al. 2008] for example use RANSAC-based isometric subgraph extraction to retrieve symmetry candidates. Their approach can handle the one or other additional feature without problems as this does not destroy the common subgraph property. If feature points are largely inconsistent across instances of some symmetry or important feature points are lacking in some instances however, detecting such symmetries anyway gets much harder. The problem is especially present for symmetries which structurally have only a low number of feature points per instance. In these cases missing feature points have an especially severe effect.

Our framework provides a technique to selectively complete feature points participating in symmetries. This can improve the results of point-feature-based symmetry detection algorithms.

Additionally, our framework is capable of providing feature-to-feature-point pairwise correspondence scores. We will describe in chapter 4.7 how this information can be used to extract feature correspondence classes in a simple way. While – due to their derivation from feature point positions only in the current implementation – the correspondence information provided is not guaranteed to be correct with respect to the underlying scene geometry (compare chapter 5), the information can be used as an initialization for subsequent full-scene symmetry detection. In the simplest form, an additional full-scene-based growing and validation step might be sufficient to extract reasonable symmetries. Such symmetry detection could profit from the robustness of the overlaying sub-symmetry approach that we describe in chapter 4.8.4.

4 Our Approach to Symmetry Feature Extraction

We present a practical way to extract approximate symmetry features for rigid symmetries. For this we introduce a general algorithmic framework to enable symmetry feature extraction and propose the idea of metric sub-symmetry detection, which due to its robustness against incomplete feature point sets is especially well suited for operating within that framework.

We start by introducing a number of notations and provide brief definitions for the terms we are using (chapter 4.1). We then give a short overview of our framework (chapters 4.2 and 4.3) and describe a method for automated propagation function applicability analysis (chapter 4.4). In part 4.5, we evaluate different data representation methods for feature evidence. Further, we describe an algorithm for feature-based correspondence class extraction in chapter 4.7 and finally give an in-depth description of our implementation of the framework, including the concept of sub-symmetry detection (chapter 4.8).

4.1 Definitions and Notations

$\Omega \subset R^3$ with $|\Omega| \in \mathbb{N}$ is a set of points which provides a point-sampling of the input scene. We assume that the position of each single point $x \in \Omega$ is subject to Gaussian noise and refer to the variance of that noise by σ^2. For simplicity reasons, this parameter is global for the whole scene. Where not denoted differently, we use $\sigma = 0.8\, a_\Omega$ throughout our experiments with a_Ω being the average point sample spacing for Ω.

We use a parameter $minStructureSize \in \mathbb{R}^+$ to denote the minimum distance between two points $x, y \in \Omega$ at which x and y must be distinguished in a structural sense. The idea behind this is illustrated by the following example: Let Ω be a scene containing a house made of bricks. It is unclear then whether one wants to find symmetries on the scale of windows, doors and the like, or on the smaller scale of individual bricks and similarly scales objects. The *minStructureSize* parameter accounts for this ambiguity. If not stated differently, our examples use $minStructureSize = 7\, a_\Omega$ with a_Ω again being the average point sample spacing for Ω.

Conceptually we consider feature points as being spherical volumetric entities in space. We use a radius of *minStructureSize* for feature point spheres. As *minStructureSize* is a global parameter, a feature point is fully characterized by its center point $\xi \in \Omega$.

$\Xi \subset \Omega \times [0,1]$ with $(\xi, \gamma_\xi), (\xi, \gamma_\xi') \in \Xi \Rightarrow \gamma_\xi = \gamma_\xi'$ is a set of feature points together with exactly one certainty score for each feature point. If not specified differently, Ξ denotes the preliminary set of feature points known at the respective time of processing, which is not necessarily the final set of feature points that the algorithm gives as its result. In contrast we use $\Xi_\infty \subset \Omega$ to denote the actual or "desired" set of feature points in a scene. While Ξ_∞ is usually unknown, it is helpful for theoretical considerations.

We use the notation (ξ, γ_ξ) for elements of Ξ, where $\xi \in \Omega$ is the center point of the feature and $\gamma_\xi \approx P(\xi \in \Xi_{act})$.

We use $G \subset [0,1] \times (\mathbb{R}^3 \to [0,1]) \times 2^\Omega \times \mathbb{R} \times 2^{\Xi \times [0,1]}$ to denote a specific set of feature guesses and the letter g for elements of such set. A *feature guess* g is a tuple of the following data:

- a certainty value $\gamma_g \in [0,1]$ representing the certainty for the feature proposed by g to be in Ξ_{act}
- a distribution $\tau_g : \mathbb{R}^3 \to [0,1]$ together with a set of relevant points $T_g \subset \Omega$ for that distribution and a normalization factor $c_g \in \mathbb{R}$ such that $c_g \sum_{x \in T_g} \tau_f(x) = 1$. For some point $x \in \Omega$, the value $c_g \tau_g(x)$ gives an approximation of the probability that

the feature proposed by g has x as its center point. In other words: $c_g \tau_g$ provides a probability distribution over the potential positions of the guessed feature point. Chapter 4.6 gives an in-depth explanation about feature guess distributions.

– optionally, a set of correspondence guesses $K_g \subset \Xi \times [0,1]$. An element $\kappa = (\xi_\kappa, \gamma_{\xi_\kappa}) \in K_g$ denotes that with a certainty of γ_{ξ_κ}, the feature point proposed by g is in correspondence to ξ_κ with respect to some symmetry in the scene.

The following kinds on functional mappings are used in the proposed framework:

– An *interest function* is a function

$$f_{\text{int}} : \Omega \to (\Omega \to [0,1])$$

It maps from Ω to a score field $S : \Omega \to [0,1]$. Within our framework, an interest function provides for each point a probability of being a feature point, based on local criteria.

– A *propagation function* is a function

$$f_{\text{prop}} : 2^{\Omega \times [0,1]} \to 2^{[0,1] \times (\mathbb{R}^3 \to [0,1]) \times 2^{\Omega} \times \mathbb{R} \times 2^{\Xi \times [0,1]}}$$

It maps from a set of certainty-connoted feature points to a set of feature guesses.

The idea behind propagation functions is to first perform a feature-based analysis of the scene's structure, and then derive from the structural information a number of guesses about where feature points could be located.

– A *selection function* is a function

$$f_{\text{sel}} : (\Omega \to [0,1]) \to 2^{\Omega \times [0,1]}$$

It selects a set $\Xi \subset \Omega \times [0,1]$ of feature points and per-feature certainty values based on evidence from a score field.

– A c*ombiner function* is a function

$$f_{\text{comb}} : 2^{\Omega \to [0,1]} \times 2^{[0,1] \times (\mathbb{R}^3 \to [0,1]) \times 2^{\Omega} \times \mathbb{R} \times 2^{\Xi \times [0,1]}} \to \Omega \to [0,1]$$

It combines a set of score fields and a set of feature guesses into a unified score field.

Details about how the different function types are used can be found in chapter 4.3. Details about specific examples of such functions and their implementations can be found in the respective parts of chapter 4.8.

4.2 Basic Idea of our Framework

Our framework directly addresses problems from both class one (noisy point data) and class two (missing local feature evidence due to local structural problems). Please refer to chapter 3.2 for an explanation of these problem classes.

Addressing these problems first of all requires knowledge about symmetries in the scene. With such knowledge, an arbitrary set of feature points can be easily converted to a valid set of symmetry features. Without knowledge about the scene's symmetry structure, asking for a valid set of symmetry features is meaningless on the other hand. While utilizing a set of symmetry features provides a good way to perform the required symmetry analysis efficiently, local evidence alone might not lead to a complete set of symmetry features due to the problems described above. This leads to a circular problem (compare figure 6): In order to get a set of symmetry features, we need information about the scene's global symmetry structure. But in order to get such information efficiently, we need a valid set of symmetry features first.

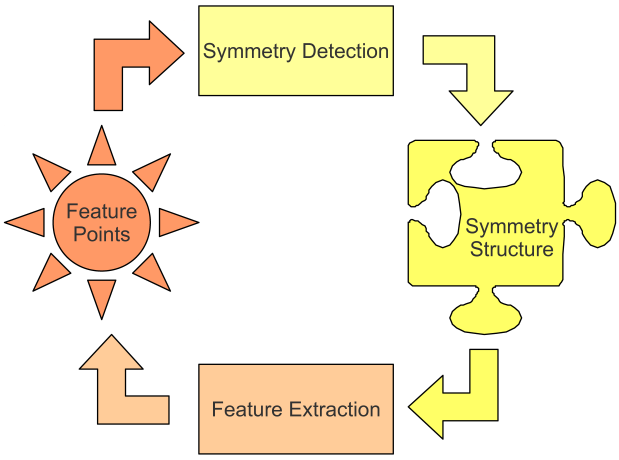

Figure 6: Dependencies in the extraction of symmetry features

For explanatory simplicity, we in this paragraph consider the set of feature points obtainable by local criteria as being a subset of all symmetry features in the scene. One can alternatively consider it a superset (requiring removing points in order to get symmetry features) or neither of both (allowing to both add and remove points from the set along the way towards a set of symmetry features). Assuming that we have a way to extract at least a subset of all partial symmetries in the scene even though local evidence might be insufficient in some instances, we can first extract those partial symmetries, and then utilize the transformation functions of these symmetries to translate feature points which are part of one instance to all other instances of the symmetry. By doing so, the set of feature points gets completed such that it becomes a set of symmetry features with respect to the partial symmetries detected so far. As not all symmetries might have been found due to incomplete local evidence, the fact that we now have gained additional knowledge about the scene in the form of new feature points, suggests an iterative approach. After completing the set of feature points to a set of symmetry features with respect to the partial symmetries found so far, the symmetry structure of the scene can be analyzed again. After that, the set of feature points can be further completed with respect to the more complete set of symmetries.

4.3 Pipeline Overview

The pipeline of our framework works by first retrieving an initial set of feature points based on local criteria. Then the symmetry structure of the scene is analyzed based on these preliminary feature points and the knowledge about global symmetries is used to improve the set of feature points. Using the improved set of features, symmetry analysis is repeated to achieve even further improvement of the feature set.

Figure 7 illustrates the proposed framework's pipeline together with an example for intermediate results at the different processing steps.

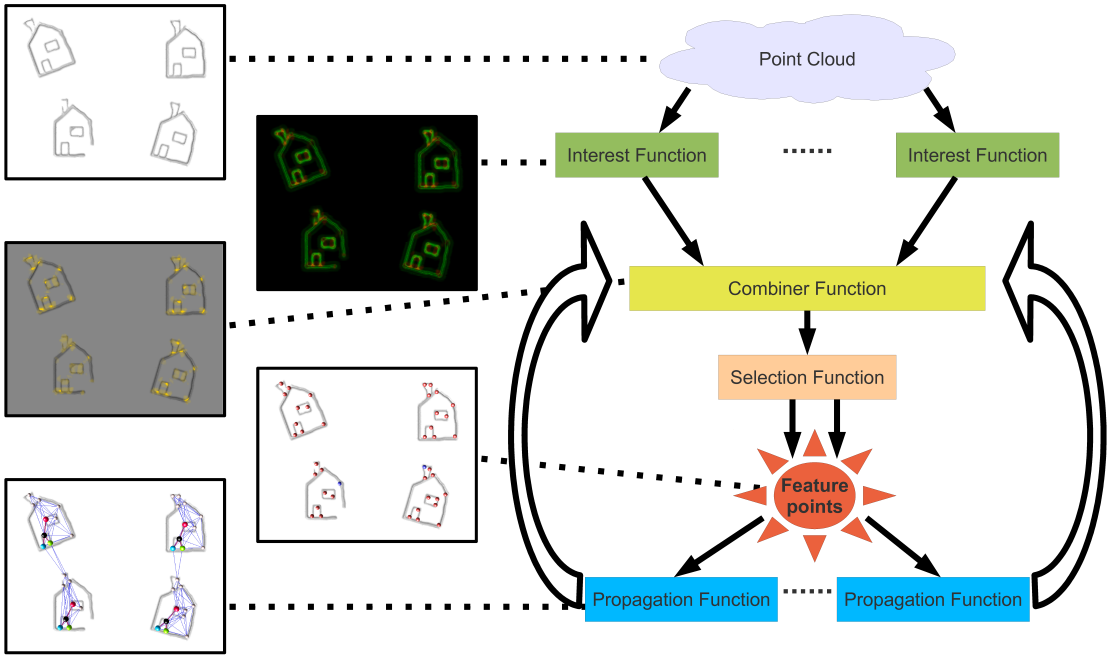

Figure 7: Symmetry feature extraction pipeline with examples for intermediate results

The following specific steps are taken within the pipeline:

1. We start with some input data $\Omega \subset \mathbb{R}^3$
2. Applying a set of interest functions $f_{\text{int},1}, \ldots, f_{\text{int},n}$ on Ω results in a number of score fields $S_1, \ldots, S_n : \Omega \to [0,1]$. Each score field tells us for each point the approximate probability that this point constitutes the center of a feature point, with respect to the respective kind of interest.
3. Before we can select a number of discrete preliminary feature points, the set of scores fields has to be combined into unified per-point scores. This is done by applying a combiner function f_{comb}. By doing so, we get a new score field $S = f_{\text{comb}}\left(\{S_1, \ldots, S_n\}, \varnothing\right)$.

4. Using a selection function f_{sel}, we can now select a first set of interest-based feature points $\Xi = f_{\text{sel}}(S)$

5. The preliminary set of features Ξ allows us to efficiently analyze the scene's global structure. We use the resulting understanding of the scene's structure to gain additional knowledge about where feature points could be located. A set of propagation functions $f_{\text{prop},1}, \ldots, f_{\text{prop},m}$ each gives us a set of feature guesses $G_1 = f_{\text{prop},1}(\Xi), \ldots, G_m = f_{\text{prop},m}(\Xi)$. We derive new sets $G_1', \ldots, G_m'$ by first applying feature guess merging (see chapter 4.5.2) on each of the sets $G_1, \ldots, G_m$ independently and then adjusting the certainties γ_g of each resulting (merged) feature guess g by multiplication with the applicability score (see chapter 4.4) of the corresponding propagation function. Finally we calculate $G = G_1' \cup \ldots \cup G_m'$ and perform a second pass of feature guess merging on the unified set G to obtain a set G' of feature guesses.

6. G' provides us with new knowledge about the positions of potential feature points. To make use of this knowledge we derive an updated score field $S = f_{\text{comb}}(\{S_1, \ldots, S_n\}, G')$ and repeat the process from step 4 on. We break out of the loop as soon as either no additional feature points are found anymore or a configurable limit on the number of iterations is reached.

4.4 Propagation Function Applicability Analysis

The applicability of a certain propagation function can differ from scene to scene. For example a propagation function which operates on rigid symmetries might not work well for a strongly deformed object while at the same time being well suited for others. Other scenes may contain a mixture of different classes of symmetries and work best with a combination of multiple propagation functions.

Our framework supports the usage of multiple propagation functions simultaneously. To make this more robust, we allow to weight each propagation functions' influence. As the optimal weighting scores are different for each scene, we perform an automated applicability analysis for each propagation function to derive a weighting factor. With this applicability analysis, propagation functions which provide feature guesses that match the features we already know due to interest well, receive a stronger weighting than those which do not fit the scene's interest features as well.

The assumption behind the analysis is that most of the designated features can be found from interest alone and only a minor number of feature points has to be completed by propagation. Based on this assumption, we will derive an expected probability value $E\left(P(\xi_g \in \Xi_\infty)\right)$ (detailed explanation follows) for each propagation function. For any feature guess $g \in G$ (with G being the set of feature guesses generated by the respective propagation function), $\xi_g \in \Omega$ in this value denotes the feature point predicted by the guess. $\Xi_\infty \subset \Omega$ is the (unknown) set of actual or "desired" symmetry features in the scene. The value $E\left(P(\xi_g \in \Xi_\infty)\right)$ allows us to estimate the relative certainty at which a randomly picked guess describes an actual feature within the given scene. Specifically, we apply a corrective factor to the certainties γ_g of all guesses $g \in G$ given by a specific propagation function (after merging) in the form that we assign new certainties $\gamma_g' = E\left(P(\xi_g \in \Xi_\infty)\right)\gamma_g$ (saying: the feature proposed by the guess is an actual feature *and* the specific guess is valid).

How can we derive $E\left(P(\xi_g \in \Xi_\infty)\right)$?

Let $\Xi_{\text{int}} \subset \Omega$ be the set of feature points detected by interest-based selection and $\xi_g \in \Omega$ and $\Xi_\infty \subset \Omega$ as defined above. For the moment, we pick a single guess $g \in G$ for explanatory reasons. An applicability value that takes the whole set G into account follows later. Please note that in this chapter we consider Ξ_∞ and Ξ_{int} without per-point certainty scores, which is in contrast to our usual definition. However we assign stochastic properties to those sets: We assume that for any given $\xi \in \Omega$, the questions whether $\xi = \xi_g$, $\xi \in \Xi_\infty$ and $\xi \in \Xi_{\text{int}}$ can only be answered with specific certainties $P(\xi = \xi_g)$ (such that $\sum_{\xi \in \Omega} P(\xi = \xi_g) = 1$), $P(\xi \in \Xi_\infty)$ and $P(\xi \in \Xi_{\text{int}})$ respectively.

We first distinguish two disjoint cases for $P(\xi \in \Xi_\infty)$:

$$P(\xi \in \Xi_\infty) = P(\xi \in \Xi_{\text{int}}) P(\xi \in \Xi_\infty | \xi \in \Xi_{\text{int}}) + P(\xi \notin \Xi_{\text{int}}) P(\xi \in \Xi_\infty | \xi \notin \Xi_{\text{int}})$$

For practical application, we approximate $P(\xi \in \Xi_\infty | \xi \in \Xi_{\text{int}})$ and $P(\xi \in \Xi_\infty | \xi \notin \Xi_{\text{int}})$ by constants. We use $P(\xi \in \Xi_\infty | \xi \in \Xi_{\text{int}}) = 0.95$ and $P(\xi \in \Xi_\infty | \xi \notin \Xi_{\text{int}}) = 0.05$ for all examples in this thesis. The fact that we can choose $P(\xi \in \Xi_\infty | \xi \notin \Xi_{\text{int}})$ as low as we do is due to our assumption of an "almost complete" interest based feature set. The term $P(\xi \in \Xi_\infty | \xi \in \Xi_{\text{int}})$ on the other hand accounts for the chance that a feature which is indicated by local interest might still not be a desirable actual feature point.

So far, we have considered $P(\xi \in \Xi_\infty)$ for a specific point ξ . The guess g however does only specify the exact location of ξ_g in the form of a probability distribution (namely τ_g). Therefore we have to consider all potential positions of ξ_g :

$$P(\xi_g \in \Xi_\infty) = \sum_{\xi \in \Omega} P(\xi = \xi_g) P(\xi \in \Xi_\infty)$$

To get the overall estimated value $E\left(P(\xi_g \in \Xi_\infty)\right)$, consider the following:

We pick a random guess $g \in G$ with

$$P(g \text{ being picked}) = \frac{\gamma_g}{\sum_{g' \in G} \gamma_{g'}}$$

This introduces a weighting which is linear in the certainty of the guesses. We use this weighting to enforce a higher influence of high-certainty feature guesses on the final applicability score compared to low-certainty feature guesses. Over this weighted "picking scheme", we estimate the expected value

$$E\left(P(\xi_g \in \Xi_\infty)\right) = \sum_{g \in G} P(g \text{ being picked}) P(\xi_g \subset \Xi_\infty)$$

which provides our applicability metric. The values of which are in the range $P(\xi \in \Xi_\infty | \xi \notin \Xi_{\text{int}}) \leq E\left(P(\xi_g \in \Xi_\infty)\right) \leq P(\xi \in \Xi_\infty | \xi \in \Xi_{\text{int}})$.

Implementation Details

Given a propagation function f_{prop}, and a set of certainty-annotated interest-based feature points $\Xi_{\text{int}} \subset \Omega \times [0,1]$ (as in our original definition for sets Ξ from chapter 4.1):

1. Generate an "overlay" mapping $o_{\Xi_{\text{int}}} : \Omega \to [0,1]$, initialized to zero everywhere, by doing the following for each feature point $(\xi, \gamma_\xi) \in \Xi_{\text{int}}$:

   Derive an updated mapping

   $$o_{\Xi_{\text{int}}}' : \Omega \to [0,1]$$

   with

   $$o_{\Xi_{\text{int}}}'(x) = 1 - \left(1 - o_{\Xi_{\text{int}}}(x)\right)\left(1 - \gamma_\xi\right) \text{ for all } x \in \Omega \,|\, \|x - \xi\| < minStructureSize$$

   and

   $$o_{\Xi_{\text{int}}}'(x) = o_{\Xi_{\text{int}}}(x) \text{ everywhere else.}$$

   Then continue with the next feature point using the updated mapping $o_{\Xi_{\text{int}}}'$ instead of $o_{\Xi_{\text{int}}}$.

   Each value $o_{\Xi_{\text{int}}}(x)$ of the generated mapping provides an estimation for $P(x \in \Xi_{\text{int}})$. Please note that we assign equal values $P(\xi \in \Xi_{\text{int}})$ for all $\xi \in \Omega$ within *minStructureSize* radius around some interest-based feature's center point in this implementation.

   The resulting mapping $o_{\Xi_{\text{int}}}$ provides a representation for $P(\xi \in \Xi_{\text{int}})$ with $P(\xi \in \Xi_{\text{int}}) = o_{\Xi_{\text{int}}}(\xi)$ for any $\xi \in \Omega$.

2. Retrieve a set of feature guesses $G_p = f_{\text{prop}}(\Xi_{\text{int}})$, apply feature guess merging (see chapter 4.5.2 for details) which results in a new set of feature guesses G

3. For each guess $g \in G$ with certainty γ_g, distribution τ_g with a discretization factor c_g and a set of relevant points T_g, calculate the sum

   $$\sum_{\xi \in T_g} c_g \tau_g(\xi) P(\xi \in \Xi_\infty)$$

   with

   $$P(\xi \in \Xi_\infty) \;=\; \Xi_{\text{int}}(\xi) P(\xi \in \Xi_{act} | \xi \in \Xi_{\text{int}}) \;+\; \left(1 - \Xi_{\text{int}}(\xi)\right) P(\xi \in \Xi_{act} | \xi \notin \Xi_{\text{int}})$$

   where $\Xi_{\text{int}}(\xi)$ is to be understood such that $\Xi_{\text{int}}(\xi) = x \Leftrightarrow (\xi, x) \in \Xi_{\text{int}}$.

   The overall applicability score for f_{prop} is then given by

   $$\frac{\sum_{g \in G} \gamma_g \sum_{\xi \in T_g} c_g \tau_g(\xi) P(\xi \in \Xi_\infty)}{\sum_{g \in G} \gamma_g}$$

4.5 Feature Guess Data Representation Methods

Both propagation functions and interest functions produce a set of guesses about where feature points may be located. While interest functions do so on a local per-data-point evidence basis, propagation functions have a more "global view" on the scene and can provide additional contextual information. Specifically, propagation functions derive their guesses based on the given set of preliminary feature data and therefore explicitly utilize dependencies between local feature probabilities. This global dependency information usually contains hints about where point to point symmetry correspondences exist.

Depending on the representation of the produced guesses for feature locations, a different degree of such contextual information is preserved for further processing. We describe two alternative guess representations in the following sections and then introduce a hybrid approach, which combines the benefits of both.

4.5.1 Implicit Guess Representation

The idea for an implicit guess representation is to operate on a score field $F:\Omega\rightarrow[0,1]$ and to incorporate the information of any single feature guess g into F as soon as possible. Whenever a guess for another feature is generated, an altered mapping $F':\Omega\rightarrow\mathbb{R}$ is derived to represent the guess. For example when considering a feature guess g with a distribution τ_g and a distribution normalization factor of c_g, we might use

$$F'(x)=1-\left(1-F(x)\right)\left(1-c_g\tau_g(x)\right)$$

as a straight-forward implementation of this approach. This equation says: x is a feature exactly if it was a feature based on the information we had available before *or* based on the new evidence provided by g.

Such an implicit representation clearly has computational advantages, as the total amount of data does not increase while incorporating additional guesses. Also, this representation allows for a straight-forward selection of feature points, as it explicitly provides scores for each point and therefore contains all information we need to decide which points to select as features.

However there are problems with this kind of representation. Obviously, using it makes us lose any information about how a certain score value $F(x)$ was composed. It therefore lacks context and information about stochastic dependencies and relation among point scores. Such information is however required for feature guess merging – which we describe below – and feature-to-feature correspondence extraction (compare chapters 4.5.2 and 4.6 respectively). An approach to solve this problem is to build an incidental mapping which maps to each point the corresponding contextual information. However this is not practical. As each feature guess can affect a large number of points, contextual information has to be duplicated across a large number of points. Still each point can be affected differently by a different set of guesses, which eliminates the possibility of sharing contextual information across points in order to eliminate the need for data duplication. Therefore, the incidental mapping can easily happen to reach memory requirements in $\Theta(|\Omega|^2)$.

4.5.2 Explicit Guess Representation

An alternative approach is to represent all feature guesses explicitly, as a set of feature guesses like defined in chapter 4.1.

Additional contextual information can be easily added to that data structure. For example we can store correspondences as a set of pairs $(\xi, \gamma_\xi) \in \Omega \times [0,1]$ with γ_ξ denoting the certainty that ξ is in correspondence to the feature proposed by the feature guess. We call this correspondence set $K \subset \Xi \times [0,1]$ and its elements $\kappa \in K$.

The explicit guess representation has the potential disadvantage of typically requiring more memory than an implicit one, with memory requirements being linear in the number of feature guesses instead of points in the scene. Whether this is actually a problem or not depends on the propagation functions used. Propagation functions should take care of limiting the number of resulting feature guesses with this representation.

An explicit guess representation also has a major computational disadvantage when one wants to retrieve per-point scores for use in the selection function. As there is no direct mapping from points to scores, retrieving a score value requires to walk over all feature guesses and combining the values of their distributions at the given point. Even though it is often possible in practice to pre-select a subset of guesses that have significant influence on a given point's score, this process is still costly when compared to the implicit representation.

Feature Guess Merging

In order to model a limited amount of dependency within a set of feature guesses, it is helpful to know which feature guesses describe one and the same feature point and which describe different ones. In practice it has turned out that to obtain useful information about the location of features from a large number of feature guesses actually requires modeling this kind of stochastic inter-guess dependency.

To understand why this is important, consider the following simplified example: Consider a set of five feature guesses with equal certainties of $\gamma_1, \ldots, \gamma_5 = 0.5$ and having equal distributions $\tau_1, \ldots, \tau_5$ which assign feature probabilities of 0.05 to a total of twenty different points $x_1, \ldots, x_{20} \in \Omega$ and a probability of 0 to all other points. Thinking of all feature guesses as describing different point features and also being otherwise independent, it is logical to assume to any point $x \in \Omega$ a probability for being a feature point of $1 - \prod_{i=1}^{5} (1 - \gamma_i \tau_i(x))$, as x is a feature as soon as any of the feature guesses is both true and says there is a feature at x. For $x_1, \ldots, x_{20}$, this leads to a certainty of

$$1 - \prod_{i=1}^{5} (1 - 0.05\gamma_i) = 1 - 0.975^5 \approx 0.12$$

Considering all five feature guesses as describing the same feature, the resulting probability for these points is

$$0.05\left(1 - \prod_{i=1}^{5} (1 - \gamma_i)\right) = 0.05(1 - 0.5^5) \approx 0.05$$

which is less than half the probability of the independent variant. Having five or more separate feature guesses for a single feature point is common with our currently implemented propagation function.

Being able to perform explicit feature guess merging is a crucial advantage of an explicit feature guess representation. As there is no information left about the underlying structure of the score field in the implicit representation case, an explicit guess merging is not possible there and can at best be approximated by some local per-point score operator with accordingly reduced precision.

We implement feature guess merging by performing the following main steps:

1. Approximate the distributions τ_g of feature guesses by a sphere with discrete center points m_g.
2. Merge together feature guesses where the corresponding center points m_g are sufficiently close to each other. We also check that at least a part of the assigned corresponding feature points matches between merging candidates as an additional merging criteria.

Complete implementation details and a discussion of our implementation's limitations follow:

Implementation Details

Given a set of feature guesses G:

1. Initialize a three dimensional KD-tree t
2. For each guess $g \in G$ with a distribution τ_g, a set of relevantly affected points T_g, a certainty of γ_g and attached correspondence information K_g:
    1. Choose a point $m_g \in T_g$ with $\tau_g(m_g) = max\{\tau_g(x) | x \in T_g\}$
    2. Validate that taking a circle of radius 3σ with center m_g gives an appropriate approximation of τ_g. More exactly, check that the ratio

$$\frac{\sum_{x \in T_g, \|x - m_g\| < 3\sigma} \tau_g(x)}{\sum_{x \in T_g} \tau_g(x)}$$

    is greater than some value, 0.5 in our experiments.
    3. If the described circle gives an appropriate approximation (see previous step):
        1. Use the KD-tree t to preselect a set of candidate feature guesses to merge g with, by finding entries $g' \in t$ with $\|m_{g'} - m_g\|^{\infty} < minStructureSize$

        While using the maximum norm favors some directions in space above others, it allows for fast candidate retrieval directly from the KD-tree. Optionally, post filtering of the results using the Euclidean norm can be performed.

        2. For each such candidate g' with a certainty of $\gamma_{g'}$ and correspondence information $K_{g'}$, assess the "average correspondence" with g by calculating

$$\frac{\sum_{(\xi, \gamma_\xi) \in K_g} \left(\frac{\gamma_\xi \gamma_\xi'}{\gamma_g \gamma_{g'}} \text{ whenever } (\xi, \gamma_\xi') \in K_{g'},\ 0 \text{ otherwise} \right)}{min\left\{ \sum_{(\xi, \gamma_\xi) \in K_g} \frac{\gamma_\xi}{\gamma_g}, \sum_{(\xi', \gamma_{\xi'}) \in K_{g'}} \frac{\gamma_{\xi'}}{\gamma_{g'}} \right\}}$$

        Then pick the candidate with the best correspondence score. Optionally continue at step 2.5 – no candidate g' exists – if the average correspondence is below some threshold for all candidates.

        Alternatively: simply pick the candidate g' with the smallest distance $\|m_{g'} - m_g\|$ (replaces step 2.3.3).

        Choose the candidate g' that has the highest correspondence matching score

        3. Merge g into this candidate (see below)
        4. If no candidate g' exists, insert g into t

Implementation Details (cont.)

4. If described circle does not gives a reasonable approximation:
    5. As we cannot easily get reliable information about how to merge such a guess, reduce its certainty to $\gamma_g * 0.05$ (or by some other constant). See below for discussion
    6. Add g to the result set
3. Add all feature guesses now in t to the result set

Merging together two feature guesses works as follows:

Given two feature guesses g_1, g_2, derive a merged feature guess g_m:

1. Set the distribution τ_m and the corresponding set of relevant points T_m of g_m to a merged distribution (see chapter 4.6.5) using the distributions from g_1 and g_2 as sub-distributions
2. Set the certainty γ_m of g_m to $1-(1-\gamma_1)(1-\gamma_2)$, meaning g_m is an applicable guess whenever g_1 and/or g_2 are applicable
3. If correspondence information is available, consider correspondences $\kappa \in K_1 \cup K_2$. For each such $\kappa=(\xi, \gamma_\xi)$ check that $\neg\exists \gamma_{\epsilon,m}$ such that $(\xi, \gamma_{\epsilon,m}) \in K_m$. If that condition is met, distinguish three cases:
    1. $\exists \gamma_{\xi,1}$ such that $(\xi, \gamma_{\xi,1}) \in K_1 \wedge \exists \gamma_{\xi,2}$ such that $(\xi, \gamma_{\xi,2}) \in K_2$:

       add to K_m the correspondence $\left(\xi, 1-(1-\gamma_{\xi,1})(1-\gamma_{\xi,2})\right)$
    2. $\exists \gamma_{\xi,1}$ such that $(\xi, \gamma_{\xi,1}) \in K_1 \wedge \neg\left(\exists \gamma_{\xi,2} \text{ such that } (\xi, \gamma_{\xi,2}) \in K_2\right)$:

       add to K_m the correspondence $\left(\xi, \gamma_{\xi,1}\right)$
    3. $\neg\left(\exists \gamma_{\xi,1} \text{ such that } (\xi, \gamma_{\xi,1}) \in K_1\right) \wedge \exists \gamma_{\xi,2}$ such that $(\xi, \gamma_{\xi,2}) \in K_2$:

       add to K_m the correspondence $\left(\xi, \gamma_{\xi,2}\right)$

While this feature guess merging algorithm works well in practice and has good performance, it has two disadvantages, which should be considered:

1. The order in which feature guesses are inserted into the KD-tree t is arbitrary. This order can is potentially relevant for the question whether two guesses get merged or not. As an example, consider feature guesses g_1, g_2, g_3 with max-points

   $m_1=(0,0,0), m_2=(1,0,0), m_3=(-1,0,0)$

Implementation Details (cont.)

Using for example $minStructureSize = 1.5$, inserting in the order of g_1, g_2, g_3 leads to exactly g_1, g_2 being merged, while inserting g_2, g_1, g_3 leads to all g_1, g_2, g_3 being merged together, as both m_1 and m_3 fit into a $minStructureSize$ radius around m_2. The proposed "average correspondence" score also is not invariant over different merge-orders.

2. The handling of guesses g with distributions not approximately representable by a circle of radius 3σ around m_g is largely arbitrary.

   The rationale behind ignoring those guesses and simply "punish" them by reducing their certainty by a constant factor is that those guesses do not give well constrained predictions anyway, but usually have a high amount of uncertainty in the associated distribution τ_g. Lowering their certainty reduces their impact on feature point selection and makes up for the potentially too high per-point feature scores that can arise by not merging a set of feature guesses , although they actually predict one and the same feature point.

   The exact factor of certainty decrease is however chosen such that works well with the current propagation and combination functions on the tested scenes only. Different functions and/or scenes might require different factors.

4.5.3 Hybrid Approach

In order to combine the advantages of low information loss as with the explicit representation with the advantage of fast per-point score feature selection, a hybrid approach can be taken. The idea is to take an explicit feature guess representation and to materialize its projection to a point-scores domain as with the implicit representation.

In order to perform a projection to per-point scores, the information of the explicit set of feature guesses must be "combined" to get a score function $F:\Omega\rightarrow[0,1]$. We use a score range between zero and one to loosely resemble probabilistic per-point scores, where for some point $x\in\Omega$, the value $F(x)$ approximately represents the probability that x is a feature point. Please note that we usually do not have $\sum_{x\in\Omega}F(x)=1$, as there can be multiple feature points in the scene. F therefore does not provide an overall probabilistic distribution with respect to feature positions. The probabilistic property is given for individual values $F(x)$ only.

The actual combination of guesses into such a score field is handled by a combiner function (details follow in chapter 4.8.2). With our combiner functions, the computational complexity of the combination process is linear in the number of feature guesses times the number of points affected by each feature guess (as given by T_g for some guess g).

One important question remains: Assuming that we have selected some point feature $\xi\in\Omega$ based on the score field. How can we go back to a set of feature guesses? More specifically: how can we find out which feature guesses participated (how much) to the set of per-point scores which led to the selection of ξ?

To tackle this problem, we can first utilize the sphere character of a feature point ξ to derive a set of points

$$P=\{x\in\Omega \mid \|x-\xi\|<minStructureSize\}$$

which conceptually belong to the feature. Then, we can walk over the set of feature guesses G and for each $g\in G$ with a certainty γ_g, a distribution τ_g and a distribution normalization factor c_g, calculate its "involvement" in ξ:

$$i_g=\gamma_g c_g\sum_{x\in P}\tau_g(x)$$

What we get is a set $I_\xi\subset G\times[0,1]$ of feature guesses and their "involvement" in ξ. The computational complexity for this is linear in the number of feature guesses, as well as in the size of P.

In all following parts, we assume a hybrid feature guess representation.

4.6 Feature Guess Distributions

A feature guess distribution $\tau_g : \mathbb{R}^3 \to [0,1]$ for some feature guess g specifies a probability density over $\mathbb{R}^3$, telling how probable it is for the feature point proposed by g to be located within different (volumetric) areas. For performance reasons, we additionally use a set $T_g \subset \Omega$ which contains only points from Ω which τ_g is relevant for. If no reasonable choice for such subset exists, we assume $T_g = \Omega$.

4.6.1 Discretizing a Continuous Distribution

Feature guess distributions provide a probability density over $\mathbb{R}^3$. However, our framework operates on a finite subset $\Omega \subset \mathbb{R}^3$. In order to apply a feature guess distribution τ to Ω, it is necessary to discretize the distribution, while maintaining the property $\sum_{x \in \Omega} \tau(x) = 1$. Conceptually this involves two things:

1. Limiting the image of τ from $\mathbb{R}^3$ to the manifold described by Ω, as we assume that feature points can only lie on the surface and not at arbitrary positions in space.
2. Discretizing the resulting continuous surface distribution to the points in Ω.

Mathematically, this process can be described as follows: We assume that any point $x \in \Omega$ implicitly describes a contiguous surface patch $\chi_x \subset \mathbb{R}^3$. Then we must find a factor $c \in \mathbb{R}$ such that

$$c \sum_{x \in \Omega} \int_{x' \in \chi_x} \tau(x') \, dx' = 1$$

We can then assign to any point $x \in \Omega$ a probability of

$$c \int_{x' \in \chi_x} \tau(x') \, dx'$$

In practice, this way of calculation gives rise to various problems however. First, we cannot easily determine a patch χ_x exactly. Second, calculating the integral over a distribution function is usually non-trivial. Especially for the case of Gaussian distributions, no analytical solution for such an integral exists in the general case. Even for simple distributions, the potentially non-uniform appearance of χ_x would make calculating the integral complicated. Therefore, we apply two simplifications:

1. We estimate $\int_{x' \in \chi_x} \tau(x') \, dx'$ by $area(\chi_x) * \tau(x)$. As long as Ω samples the surface densely compared to the standard deviation of the distribution, this provides a feasible approximation for most types of practical distributions.

2. Instead of calculating χ_x and therefore $area(\chi_x) = \int_{x' \in \chi_x} dx'$ exactly, we assume a uniform sampling of the surface. In other words: $area(\chi_x)$ is (approximately) the same for each $x \in \Omega$. In practice, most distributions are locally constrained, meaning that a comparably small subset of points $T \subset \Omega$ accounts for most of the sum $\sum_{x \in \Omega} \tau(x)$. With this, we can lower our assumption of a globally uniform sampling to assuming only a "locally" uniform sampling over T.

Applying both simplifications, the constant c can now be numerically calculated as

$$c = \left(\sum_{x \in \Omega} \tau(x) \right)^{-1}$$

Sacrificing some additional precision, the number of calculations to perform can be further lowered by summing over just the set $T \subset \Omega$.

In the following part, we describe a number of different distributions that we are using within our implementation.

4.6.2 Gaussian Dot Distribution

The *Gaussian dot distribution* describes the probabilities for all possible positions of a single point in space, given an estimated position, under the assumption that the derivation from the estimated position is normally distributed and uniform for each direction in space (i.e. no direction is "favored" by the estimation error over another direction). As long as the underlying sources of the error in estimating the point position are not known, assuming a normal distribution is reasonable, as it describes the case of summing over an infinite number of pairwise-independent sources of error (central limit theorem).

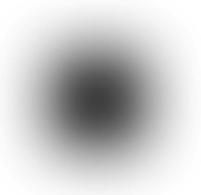

Figure 8: Gaussian dot distribution (2D case)

Given the point position estimation ("center") $c \in \mathbb{R}^3$ and the variance for the distribution σ^2, the Gaussian dot distribution is given by a multivariate Gaussian function:

$$\tau(x) = e^{\frac{-\|x-c\|^2}{2\sigma^2}} \left((2\sigma^2 \pi)^{\frac{3}{2}} \right)^{-1}$$

As a property of the normal distribution, the actual point lies within 2σ of the estimation with more than 95% certainty. Therefore a – for our requirements – reasonable set $T \subset \Omega$ for the Gaussian dot distribution is

$$T = \{ x \in \Omega \mid \|x-c\| \leq 2\sigma \}$$

4.6.3 Gaussian Radius Sphere Distribution

With our current implementation of the propagation function, information about the position of a feature guess is known in the form of a certain distance from another reference point. Both the exact distance as well as the position of the reference point however are known up to some certainty only. The *Gaussian radius sphere distribution* describes this by modeling the distribution for a normally distributed distance from a reference point with a normally distributed error in its position. As both sources of errors behave equally in each spatial direction and are both normally distributed, the uncertainty in the position of the reference point can be seen as just another uncertainty in the distance from its estimated position, in other words: as a summation of two random variables. The variance of the combined error is simply the sum of the radius and the point position variance.

Figure 9: Gaussian radius sphere distribution (2D case)

Formally, the desired distribution can be obtained by convolving the surface of a sphere (for the 3D case) with Gaussian dot distributions and multiplication with some scaling factor in order to normalize the distribution's integral to one. For our needs, a simpler approach is feasible. For some reference point $c \in \mathbb{R}^3$, an estimated distance $r = \mathbb{R}$ and a combined error variance σ^2, we describe the Gaussian radius sphere distribution by the term

$$\tau(x) = e^{\frac{-(\|x-c\|-r)^2}{2\sigma^2}} \left((2\sigma^2\pi)^{\frac{1}{2}} 4\pi r^2 \right)^{-1}$$

This term describes a normal distribution over $\|x-c\|-r$ normalized by the surface area $4\pi r^2$ of a sphere.

Intuitively speaking, there are two errors that this simplification implies:

1. Towards the center of the sphere it gives a smaller probability than would be achieved by the convolution of Gaussian dot distributions, while for the outside of the sphere (for $\|x-c\| > r$), the values given are too high. What one would expect when integrating over the inside and outside volume of the sphere defined by the center point c and the radius r is an equal share of probability for the actual point to be inside as well as outside of it. However the given simplification does not correctly parametrize over the volume, but instead represents a parametrization over the radials from the estimated center point.

2. Around the center of the sphere, the values given by τ are smaller than expected also due to another principal error: the normal distribution over $\|x-c\|-r$ gets truncated at $\|x-c\| = 0$ with our simplification

Due to these errors, an integral value $\int_{\mathbb{R}^3} \tau(x)\,dx < 1$ is to be expected. While this normalization problem is neutralized by the renormalization that we perform when discretizing the distribution to Ω (see above), the relative error in the distribution remains. With the estimated distance r being sufficiently large compared to the combined standard deviation σ, this error is minor however.

As with the Gaussian dot distribution, we can additionally cut of points that receive a cumulated probability of less than 5 % by using

$$T = \{ x \in \Omega \mid -2\sigma \leq \|x - c\| - r \leq 2\sigma \}$$

4.6.4 Multiplicative Distribution

The *multiplicative distribution* models the case where multiple sub-distributions are to be overlayed in a logical *and* fashion. Given a set of sub-distributions $\tau_1, \ldots, \tau_n$, the corresponding multiplicative distribution is given by

$$\tau(x) = \prod_{i=1}^{n} \tau_i(x)$$

As $\tau_i(x) \leq 1 \; \forall \, 1 \leq i \leq n, x \in \mathbb{R}^3$, it is $\int_{x \in \mathbb{R}^3} \tau(x) \, dx < 1$ in most cases. The multiplicative distribution is therefore not strictly a continuous probability distribution. As with the simplified Gaussian radius distribution, the discretization step however eliminates this problem.

Figure 10: Multiplicative distribution (2D case, white); Two Gaussian radius sphere sub-distributions (yellow) shown here for better understanding

Given sets $T_1, \ldots, T_n \subset \Omega$ for the sub-distributions, we use

$$T = \bigcap_{i=1}^{n} T_i$$

to restrict calculations to significant points only.

4.6.5 Merged Distribution

The *merged distribution* represents the combination of two sub-distributions which are assumed to describe the same feature point.

Figure 11: Merged distribution (2D case); Two Gaussian radius sub-distributions

To see why we implement this distribution the way we do, first consider a set of sub-distributions $\tau_1, \ldots, \tau_n$ instead of just a pair of sub-distributions. Considering $\tau_1, \ldots, \tau_n$ as describing the same actual feature point, a multiplicative distribution would be the first choice to represent the joint probabilistic feature position distribution, as all sub-distributions have to agree on a specific feature position. However, just one single error in the decision whether two distributions describe the same feature or not can effectively destroy such a multiplicative distribution numerically as well as semantically. For example consider the extreme case of two distributions following an approximate Dirac Delta function. Erroneously merging those functions multiplicatively results in an ill-defined distribution. The same problem exist in a less dramatic but still relevant way for most practical distributions, making a meaningful discretization of the resulting distribution numerically impossible.

In theory, this problem can be accounted for by explicitly handling the case of an erroneous merging. For example for two sub-distributions τ_1, τ_2 one could calculate a joint distribution as a combination of two disjunct cases:

1. With a probability of $\varepsilon \in [0,1]$, τ_1, τ_2 were merged correctly. In this case we use a multiplicative distribution with values $v\,\tau_1(x)\tau_2(x)$ for any point $x \in \mathbb{R}^3$, where υ is a normalization factor which satisfies $\upsilon \int_{x \in \mathbb{R}^3} \tau_1(x)\tau_2(x) = 1$
2. With a probability of $1-\epsilon$, τ_1, τ_2 were merged incorrectly. In this case we derive values for our distribution by giving probability values which describe a logical *or* operation: $1-(1-\tau_1(x))(1-\tau_2(x))$

Combining both cases results in

$$\tau(x) = \varepsilon\,\upsilon\,\tau_1(x)\,\tau_2(x) + (1-\varepsilon)\left(1-(1-\tau_1(x))(1-\tau_2(x))\right)$$

While this works for just two merged distributions, it requires to consider all of the 2^{n-1} possible combinations of correct and erroneous merges when considering n sub-distributions.

A more efficient alternative is to implicitly model in the possibility of an error. Consider a distribution which merges two sub-distributions τ_1, τ_2. The idea is to establish a "base certainty level" for each distribution, in order to reduce the damage that results from erroneous merging. Let γ_1, γ_2 be the certainties of the feature guesses that τ_1 and τ_2 belong to. Let ε be the certainty that the merging decision was correct (we use a constant value of 0.8 for ε in our experiments). For each of the two sub-distributions, we distinguish between the case that both the corresponding feature guess *and* the merging are correct, and the case that either one is not. This leads to certainty values of $\gamma_1 \varepsilon \tau_1(x) + 1-\gamma_1 \varepsilon$ and $\gamma_2 \varepsilon \tau_2(x) + 1-\gamma_2 \varepsilon$ respectively. Multiplying those values yields $(\gamma_1 \varepsilon \tau_1(x)+1-\gamma_1 \varepsilon)(\gamma_2 \varepsilon \tau_2(x)+1-\gamma_2 \varepsilon)$. These values unfortunately have the problem that we get

$$\int_{x \in \mathbb{R}^3} (\gamma_1 \varepsilon \tau_1(x)+1-\gamma_1 \varepsilon)(\gamma_2 \varepsilon \tau_2(x)+1-\gamma_2 \varepsilon)\,dx \;\geq\; \int_{x \in \mathbb{R}^3} (1-\gamma_1 \varepsilon)(1-\gamma_2 \varepsilon)\,dx \;=\; \infty$$

As a fix for this issue, we limit the value at any point x to the larger one of $\tau_1(x), \tau_2(x)$. This finally gives the overall formula for the merged distribution:

$$\tau(x) = min\left(max\left(\tau_1(x), \tau_2(x)\right), (\gamma_1 \varepsilon \tau_1(x)+1-\gamma_1 \varepsilon)(\gamma_2 \varepsilon \tau_2(x)+1-\gamma_2 \varepsilon)\right)$$

To generalize from the case of two sub-distributions to the general case of n sub-distributions, the pairwise merged distribution can be applied recursively.

We use

$$T = T_1 \cup T_2$$

for limiting calculations to relevant points only.

4.7 Feature-to-Feature Correspondence Extraction

Feature guesses may carry information about the underlying feature-to-feature correspondences. This information can be used to extract a set of feature correspondence classes. Such correspondence classes can be used as a starting point for the extraction of symmetries in Ω for example. Figure 12 shows an example of extracted correspondence classes.

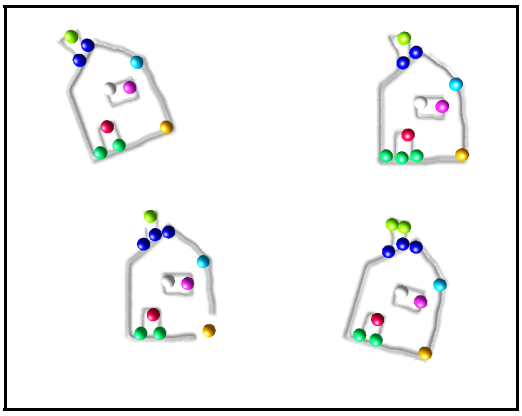

Figure 12: Feature correspondences; Large spheres denote corresponding features, one color (including white) per correspondence class

Given a set of feature-to-feature correspondence certainties $\Xi\times\Xi\rightarrow[0,1]$, extracting correspondence classes can be formulated as a graph problem. The idea is to identify connected components in a graph over all feature points, where two feature points are connected exactly if they are in correspondence to each other. The following gives an algorithm to perform correspondence class extraction:

1. Build a weighted undirected completely connected graph G, with one node per feature point. Initially, all edge weights are one. The weight of an edge e between two features ξ_1,ξ_2 in G represents the certainty that ξ_1,ξ_2 are *not* in correspondence to each other.
2. For each feature point $\xi\in\Xi$:
    1. Retrieve the set of underlying feature guesses $I_\xi\subset G\times[0,1]$ as described in chapter 4.5.3
    2. For each pair $(g,i_g)\in I_\xi$ (where g is one of the underlying guesses and i_g tells how strong g's influence in the selection of ξ was), consider the set of correspondences K_g. For each $\kappa=(\xi_\kappa,\gamma_{\xi_\kappa})\in K_g$, update the weight of the edge between ξ and ξ_κ by multiplying its current weight with $1-i_g\gamma_{\xi_\kappa}$. Semantically this means that two features ξ_κ and ξ are *not* correspondent exactly if they were *not* corresponding before *and* they are *not* corresponding according to the new evidence provided by κ.
3. Extract an unweighted undirected auxiliary graph G' with one node per feature point and an edge between two features ξ_1,ξ_2 whenever the weight of the corresponding edge in G is below one minus some certainty threshold. In all examples, we use a certainty threshold of 0.3.
4. Using G', extract connected components. Each component provides a correspondence class.

4.8 Instances of Framework Functions and their Implementation

In this part, we present concrete instances for interest, combiner, selection and propagation functions as well as information about our implementation of those functions.

We implemented the framework in C++ within the XGRT (http://www.gris.uni-tuebingen.de/xgrt) toolkit. XGRT provides data structures and basic algorithms to work on three dimensional point clouds efficiently. It also provides rendering facilities and tools for interactive point cloud editing. We utilize the provided data structures of XGRT for operating on point clouds and plug into XGRT's user interface. The user interface itself is based on the Nokia Qt toolkit, which enables compilation on a range of different platforms.

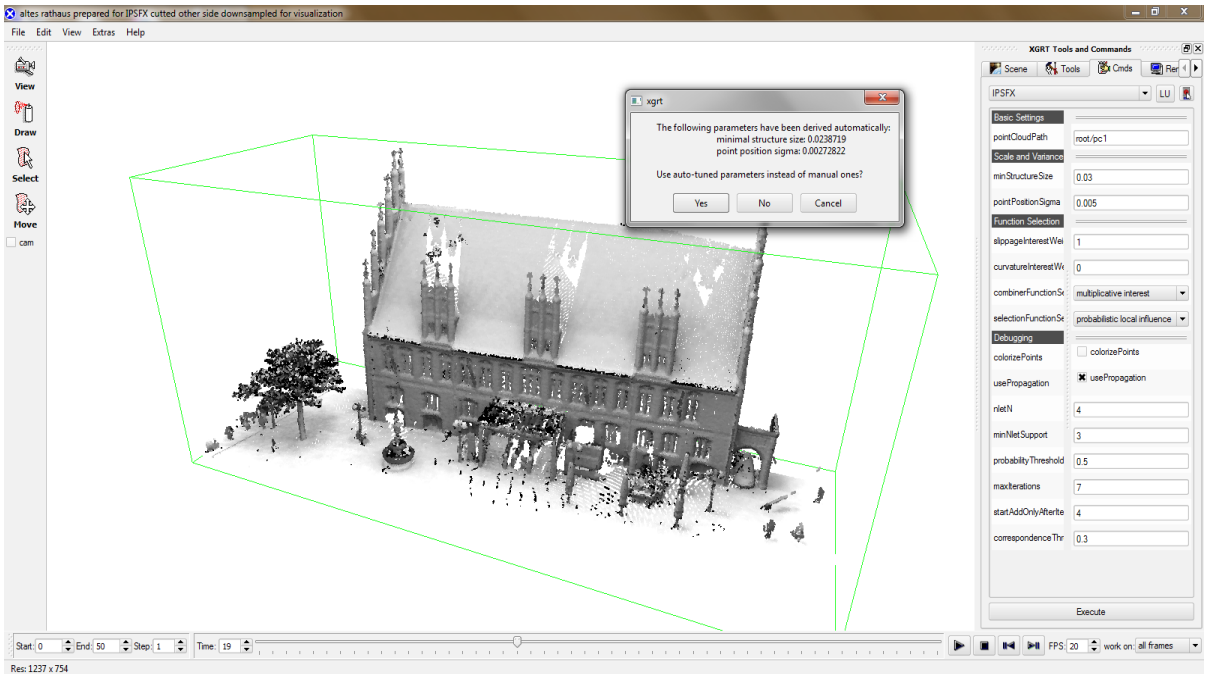

Figure 13: XGRT user interface

All tests and experiments have been performed on multi-core 64 bit AMD64 systems running Microsoft Windows XP or Windows 7. Exact timings for a specific hardware environment are given in chapter 5. The XGRT toolkit as well as the framework and function implementations have been compiled using the Microsoft Visual C++ 2008 compiler.

4.8.1 Interest Functions

Given the point cloud $\Omega \subset \mathbb{R}^3$, an interest function gives a functional mapping $S : \Omega \rightarrow [0,1]$. For some point $x \in \Omega$ the value $S(x)$ should be loosely interpretable as the probability that x is a feature point, when judging from the specific interest information solely.

For the demonstration of our framework, we have implemented two different interest functions, which use different local criteria to produce S. First we give an overview over slippage analysis as proposed in [Gelfand et al., 2008] and describe a slippage-based interest function. Secondly, we introduce an interest function which uses interest based on Gaussian curvature.

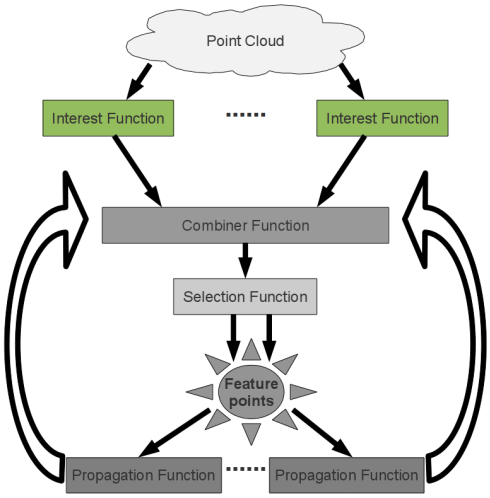

Figure 14: Interest function

Slippage Interest Function

Slippage analysis states the problem of how well the alignment of some local piece of geometry to itself is constrained. Mathematically this works by considering a quadratic least squares energy function. The Eigenvectors with zero-Eigenvalues (in practice: near-zero Eigenvalues) of the Hessian of this function then describe motions, which can be performed without increasing the value of the energy function on an infinitesimal scale. In other words, a local alignment of the patch to itself is not constrained in this direction. Such a motion is called a "slippable motion". Figure 15 illustrates the intuitive meaning of slippable translations.

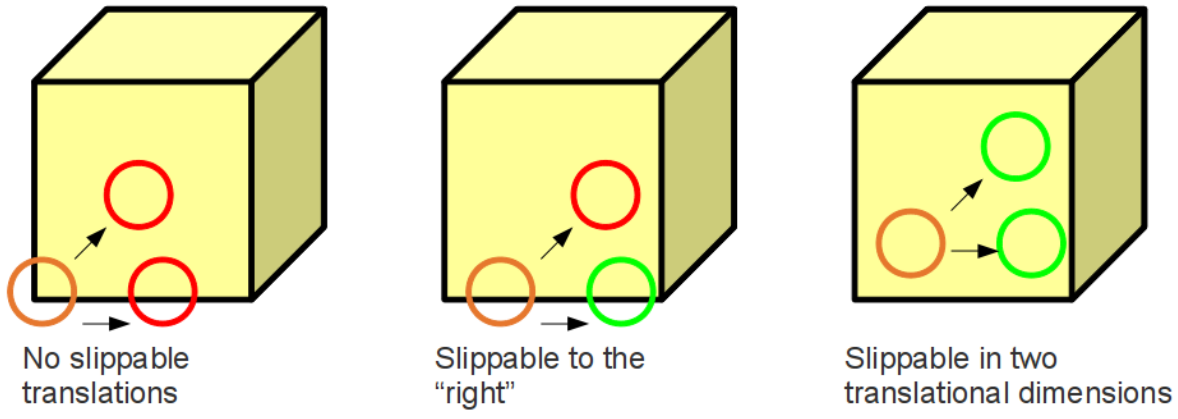

Figure 15: Intuition for slippable motions: considering a point in the orange circle, some translations (red) are well constrained by the point's local environment, others less so (green)

Given some point, local slippage analysis is performed by first building a quadratic energy function over a neighborhood, where the energy function describes how well the point's neighborhood aligns to itself under different transformations. The influence of points in the neighborhood is weighted by a Gaussian function. Then, an Eigenvector decomposition of

the corresponding Hessian matrix is performed. We use an existing implementation from [Bokeloh et al. 2008] for performing slippage analysis, which is based on [Gelfand et al. 2008]. For the Gaussian neighborhood weighting, we use a standard deviation of $\sigma = 0.5 * minStructureSize$. Using the ratio between the smallest and largest eigenvalue of the Hessian gives a non-slippability score, where a value near zero indicates slippability in at least on direction (rotational and/or translational) while a score value of nearly one indicates a well constrained auto-alignment problem. Cases where exactly one slippable motion exist can be detected by additionally considering the second largest eigenvalue. Figure 16 shows an example for the results of slippage analysis.

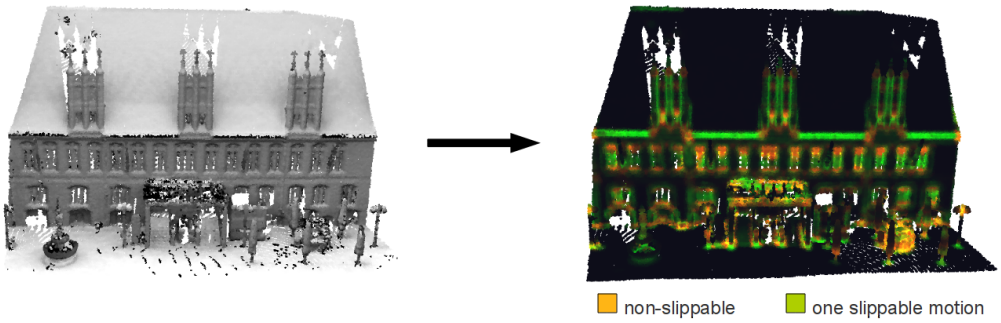

Figure 16: Results of local slippage analysis (Old Hannover town hall data set)

For extracting point features, non-slippable points are most interesting obviously. However the exact values of the described non-slippability score can vary depending on the choice of $minStructureSize$, the overall structure of the scene as well as the quality of the point sampling. To minimize the effects of different scalings, we derive a rescaled preliminary score mapping $S' : \Omega \rightarrow [0,1]$ such that the upper percentile of points with respect to non-slippability receives a score of at least 0.85 and points with below-median interest get a score value of 0. The mapping S' is derived as follows:

1. We assign to each point $x \in \Omega$ a non-slippability score

   $$s_x = slip_0(x) + 0.005\, slip_1(x)$$

   where $slip_0(x)$ is the ratio between the smallest and largest eigenvalue of the Hessian and $slip_1(x)$ the ratio between the second smallest and largest eigenvalue. We incorporate $slip_1(x)$ into the score since points with no more than a single slippable motion should still receive a small amount of interest. Especially when combining the resulting slippage-based interest values with additional evidence later in the framework, the addition of a small fraction of $slip_1(x)$ helps in making feature points "snap" to edges in the scene. The weighting factor of 0.005 has turned out to work well with all tested data sets.

2. By sorting all scores s_x, we determine the median value β as well as a lower bound for the upper percentile of all point's non-slippability scores $s_{\text{u. p.}}$.

We calculate

$$\alpha = \frac{\frac{1}{1+0.85} - 1}{max\left(0, |s_{\text{u. p.}}| - \beta\right)} \quad *$$

3. Define

$$S'(x) = 1 - \frac{1}{\alpha \, max\left(0, |s_x| - \beta\right) + 1}$$

for any $x \in \Omega$. Please note that this specifically results in $S'(x) \geq 0.85$ for any point with s_x being in the upper percentile and $S'(x) = 0$ for points that have s_x in the lower half of all points.

* This formula follows from the scaling formula used in the definition of S' in step 3, by setting a "target score" $S'(x) = 0.85$:

$$\begin{aligned}
& S'(x) = 1 - \frac{1}{\alpha \, max\left(0, |s_x| - \beta\right) + 1} \\
\Leftrightarrow\ & 1 + S'(x) = \frac{1}{\alpha \, max\left(0, |s_x| - \beta\right) + 1} \\
\Leftrightarrow\ & \frac{1}{1 + S'(x)} = \alpha \, max\left(0, |s_x| - \beta\right) + 1 \\
\Leftrightarrow\ & \frac{\frac{1}{1 + S'(x)} - 1}{max\left(0, |s_x| - \beta\right)} = \alpha
\end{aligned}$$

Formally S' provides a feasible choice for the final interest mapping S. However it still has a major semantic problem: For each point $x \in \Omega$ the value $S(x)$ is supposed the give approximately $P(x \in \Xi_\infty)$. The values of S must be dependent on the sampling density of the point cloud. For instance doubling the number of sample points should effectively cut the individual per-point probabilities for being a feature point into half, as the number of potential feature point positions is doubled. The underlying problem here is that local per-point slippage analysis lacks a notion of discrete feature points. Specifically we would expect all per-point scores which describe a set of possible center points for a single semantic feature point to sum up to no more than one.

To encounter this inherent problem of per-point scores, we rescale the score values as follows: For each point $x \in \Omega$, we calculate

$$c_x^{-1} = \sum_{y \in \Omega, \|y-x\| < minStructureSize} 1$$

essentially calculating a local point density. During local slippage analysis, we used a standard deviation $\sigma = 0.5\, minStructureSize$. Depending on the value of σ, a single non-slippable point induces high non-slippability scores in a differently sized "area of impact" around itself. We want to ensure that for a single non-slippable point, the overall non-slippability score is no more than one. For this reason, we sum over points within a radius of $minStructureSize$ for retrieving c_x^{-1}, to approximately compensate for that effect. Instead of summing over the number one, more sophisticated weighting schemes can optionally be applied. Then, we define

$$S(x) = c_x S'(x)$$

to get the final result mapping of the slippage interest function.

Curvature Interest Function

The curvature-based interest function works just like the slippage-based interest function, but instead of using local slippage, local point scores are derived as Gaussian curvature values with a small part of principal curvature added. We only consider positive Gaussian curvature, as we do not want saddle points to receive high scores. Given the two principal curvature values $\kappa_{max}, \kappa_{min} \in \mathbb{R}$, the per-point score gets calculated as

$$s_x = max\{0, \kappa_{max}\kappa_{min}\} + 0.007|\kappa_{max}|$$

where the factor 0.007 – similar to the slippage case – is meant to also assign some small score value to "edge points". Scaling this score to probability values then works exactly as for the slippage interest function.

For Calculating principal curvature values, we are using an existing implementation from the XGRT framework. It works by first fitting a local PCA coordinate frame using a Gaussian weighted neighborhood of the current point and then fitting a quadratic polynomial to the points in this neighborhood. The Eigenvalues of the corresponding Hessian matrix provide the principal curvature values. For the Gaussian weighting of the neighborhood, we use $\sigma = 0.5\, minStructureSize$.

4.8.2 Combiner Functions

A combiner function maps from a set of feature guesses G and a set of interest score fields $S_1, \ldots, S_n : \Omega \to [0,1]$ to a unified probability field $F : \Omega \to [0,1]$. F loosely specifies for each single point in the scene, how probable it is for this point to be a feature.

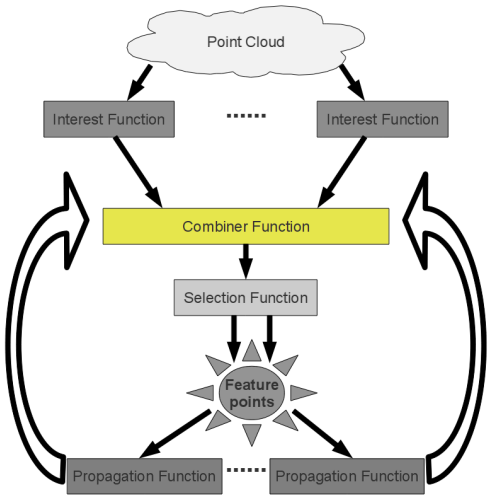

Figure 17: Combiner function

The following example gives an intuition what a combiner function does: Let us say that a propagation function resulted in a feature guess which tells that there is a feature point somewhere on the surface of a sphere with radius r around a point $p \in \Omega$. Let us further assume that we are 50% certain that this guess is correct. Then – assuming a uniformly sampled point cloud – the per-point feature-probability is $0.5\,n^{-1}$ for each of the $n \in \mathbb{N}$ points on the sphere's surface when judging from this guess alone. However, we might have additional information available. Say we have a second guess that predicts a feature somewhere on the surface of a sphere of radius r' around another point $p' \in \Omega$ with $\|p - p'\| \leq r + r'$ at a certainty of 70%. Let $n' \in \mathbb{N}$ be the number of points on that sphere's surface. The constellation of these two feature guesses is illustrated in figure 18. With this additional evidence for a potential feature point, the resulting probability for points which are on both surfaces should receive an increased feature-probability value when comparing to the values resulting from each of the two guesses alone. For completely independent predictions, the intersecting points should get a feature-probability of $1-\left(1-0.5\,n^{-1}\right)*\left(1-0.7\,n'^{-1}\right)$. This formula represents that a point is a feature, whenever it is a feature with respect to at least one of the two predictions. If there is a stochastic dependency between both probability distributions, matters can get arbitrarily complicated. For practical reasons we assume all stochastic attributes of any two feature guesses to be completely independent. We rely on feature guess merging to cover the case of multiple feature guesses describing the same actual feature point (compare with chapter 4.5.2).

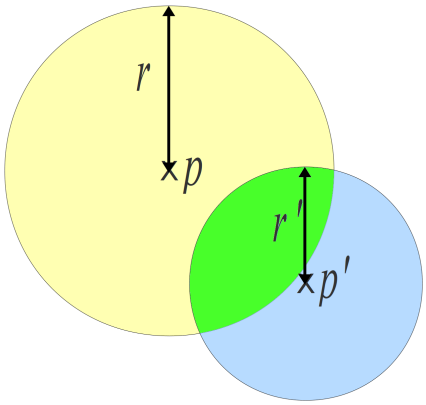

Figure 18: Overlapping feature guesses (schematic). Points in the green area should receive elevated feature-probability values.

From these considerations, the following combiner function results:

Given a set of feature guesses G and a set of $S_1, \ldots, S_n : \Omega \to [0,1]$, the resulting score field $F : \Omega \to [0,1]$ is defined by

$$F(x) = 1 - \prod_{i=1}^{n} \left(1 - S_i(x)\right) \prod_{g \in G} \left(1 - c_g \tau_g(x)\right)$$

for any $x \in \Omega$.

A consequence of this approach is that we have $F(x) \geq 1 - \prod_{g \in G} \left(1 - c_g \tau_g(x)\right)$, which means that feature guesses alone can result in a high combined per-point score even though no interest-based evidence for that point is present at all. While this is necessary to recover from errors of class two (symmetry features which lack local evidence due to structural reasons, compare chapter 3.2) it can also result in undesired "overshooting", as figure 19 illustrates.

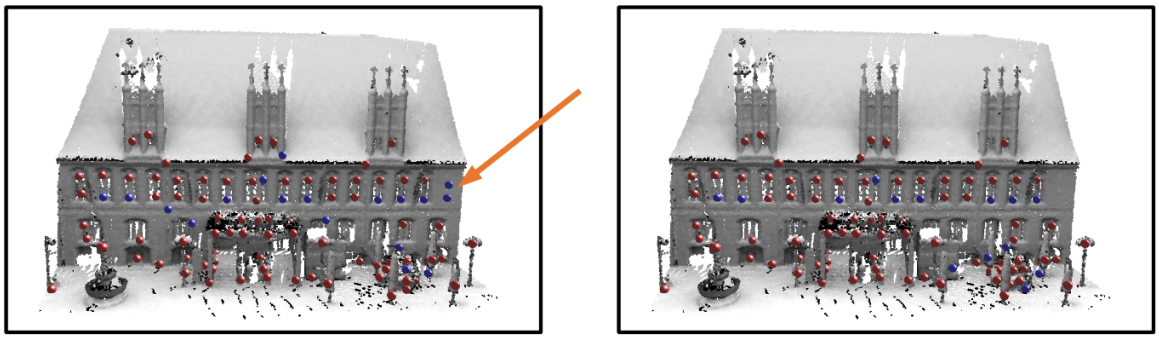

Figure 19: Comparison of combiner functions: Left without interest requirement, right with multiplicative interest

Some scenes contain only errors of class one, meaning that every symmetry feature is in principle detectable by local criteria, but local evidence might still be too low at some of the desired feature point's to make those points selected as features (due to noise etc.). Such scenes allow for another combiner function that does not suffer from the "overshooting" problem. This combiner function works by requiring at least some local evidence for any point in order to assign a high score value to that point. This *multiplicative interest combiner function* has the advantage of sticking closer to what the data provides, when compared to the previous "plain" combiner function. We numerically implement the "at least some local evidence" criteria by deriving a rescaled joint interest score field $S' : \Omega \to [0,1]$ as follows:

$$S'(x) = \left(\frac{1 - \prod_{i=1}^{n} \left(1 - S_i(x)\right)}{\max_{y \in \Omega} \left(1 - \prod_{i=1}^{n} \left(1 - S_i(y)\right) \right)} \right)^{\frac{1}{10}}$$

The overall formula for the resulting score field of the multiplicative interest combiner function follows from the idea that a point is a feature point, whenever either sufficient local interest exists, or at least some local interest exists *and* there is additional evidence from feature guesses:

$$F(x) = 1 - (1 - S'(x)) \left(1 - S'(x) \left(1 - \prod_{g \in G} (1 - c_g \tau_g(x)) \right) \right)$$

4.8.3 Selection Functions

A selection function is a functional mapping $f:(\Omega \to [0,1]) \to 2^{\Omega \times [0,1]}$. Based on a score field coming from the combiner function, it gives a discrete set of feature points together with a certainty for each selected feature point.

The specific selection function we are using selects feature points of maximal support in an iterative way. The support for a certain point $x \in \Omega$ is calculated by aggregating the provided score values for all points within a $minStructureSize$ radius around x.

Given a score field $F:\Omega \to [0,1]$, the algorithm works as follows:

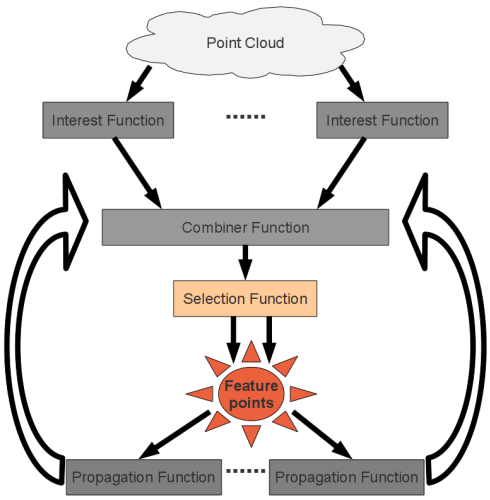

Figure 20: Selection function

1. Initialize an overlay factor field $o:\Omega \to [0,1]$ with $o(x)=1 \forall\, x \in \Omega$
2. Perform (low-pass) frequency filtering: for each point $x \in \Omega$ calculate an aggregated score

   $$a_x = \sum_{x' \in \Omega, \|x'-x\| < minStructureSize} o(x')F(x')$$

   This way of filtering involves a spherical-plateau filtering kernel, more specifically a kernel function which is one for each point inside of a sphere around x and zero everywhere else. While it is not obvious which kernel to use for frequency filtering, the spherical-plateau kernel has the advantages of being simple and fitting naturally to the idea of considering feature points as spheres.
3. Select a point $m \in \Omega$ such that $a_m = max\{a_x \mid x \in \Omega\}$
4. If a_m is smaller than some selection certainty threshold (0.5 was used for all examples), stop
5. Add (m, a_m) to the result set
6. As all underlying points $x' \in \Omega \mid \|x'-x\| < minStructureSize$ are now conceptually assigned to the newly selected feature m, those points shall not at the same time provide support for another feature point. We reflect this by effectively lowering the scores for those points to zero though.

   For this, derive a new overlay field $o':\Omega \to [0,1]$ with

   $o'(x)=0$ for all $x \in \Omega \mid \|x-m\| < minStructureSize$

   and

   $o'(x)=o(x)$ everywhere else
7. Iterate: Go back to step 2 using the updated overlay o' instead of o

A question which remains is why aggregated / frequency filtered scores a_x are necessary for the selection of features. To make the advantage of this approach clear, consider an alternative selection function which simply selects feature points based on maximum value $o(m)F(m)$. While this is simpler and significantly faster computationally, it has significant disadvantages. Figure 21 demonstrates two major problems of this approach for the simplified case of a one-dimensional score field.

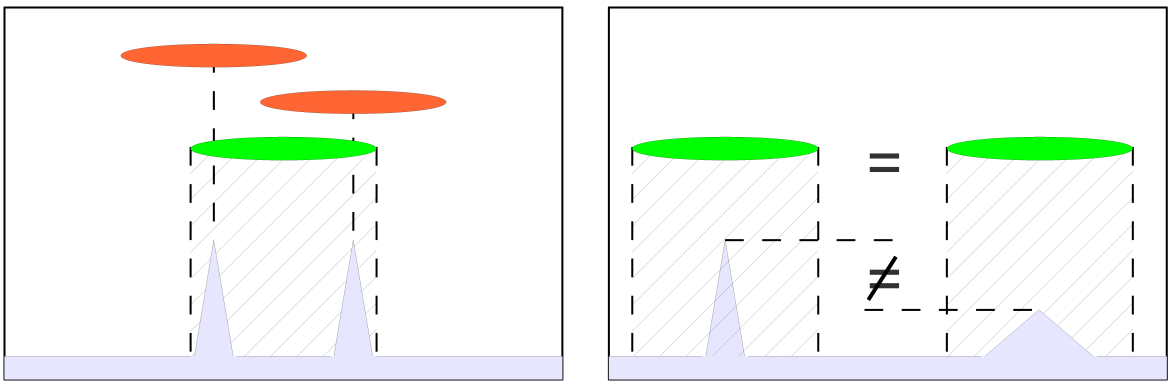

Figure 21: Frequency filtered versus local maximum selection; Left: Two peeks in the score field below the frequency limit result in the green feature for frequency filtered selection. Local maximum-based selection results in two off-center features (red), showing that high-frequency noise can badly influence local maximum based selection; Right: Spherical-plateau-based frequency filtering gracefully handles differences in feature position variance up to some degree. Local maximum selection tends to neglect feature guesses of slightly higher variance, even if the total certainty of such is higher.

The described selection function algorithm can be slightly modified to implement “add-only” selection. As described until here, each iteration in our framework includes a completely new selection of feature points. While it is often desirable to allow feature points from a previous selection to slightly move to a better location (especially one which is more consistent across instances of some symmetry), there can be other cases where it is more favorable to stick to the feature points from a previous iteration while only adding new feature points to the current set. Especially, this improves the convergence of the feature point set over a number of iterations, as no oscillation between different feature point positions is possible. For all examples, we ran the first few iterations (usually four) with feature sets selected from scratch each time, and then switched over to “add-only” iterations in order to achieve convergence.

Performing “add-only” selection with the given selection function works by pertaining both the overlay field o as well as the set of selected feature points Ξ from the previous iteration. Any new feature selected gets added to the existing set Ξ, while the overlay o ensures that no existing feature point gets selected a second time.

4.8.4 Propagation Functions

A propagation function assigns to a set of feature points Ξ a set of feature guesses. Conceptually, any propagation function has to perform the following steps:

1. Use the provided set of feature points Ξ to get an understanding of the scene's structure with respect to symmetries in the scene.
2. Based on the structural information about the scene, derive guesses where feature points should be located in order to get a set of symmetry features with respect to the symmetries detected in the previous step.

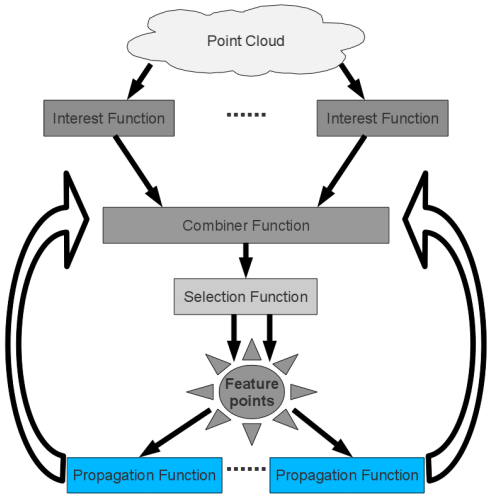

Figure 22: Propagation function

Propagation functions are the most crucial element in our approach to symmetry feature extraction. Especially, the choice of the selection function(s) constrains the class of symmetries for which symmetry features can be extracted by our framework. A propagation function which detects only translational symmetries in step one for example, cannot be used to recover missing symmetry feature points with respect to rotational symmetries.

We restrict our considerations to rigid symmetries. For this, we propose a class of propagation functions which detect symmetries of a fixed size $n \in \mathbb{N}$ in the set of feature points. We call these small-scale symmetries *sub-symmetries*. After sub-symmetries have been detected, we consider the feature-neighborhoods of all instances of a specific sub-symmetry. Features which exist in the majority of such neighborhoods lead to the generation of associated feature guesses. For the instances which lack a corresponding feature point in their neighborhood, these feature guesses allow to recover the missing features.

In the following, we first introduce the notion of sub-symmetries in more detail and propose a sub-symmetry detection algorithm. We further describe how feature guesses can be generated based on sub-symmetry information. Finally we give a complexity analysis with regard to the result data size of the proposed class of propagation functions.

Sub-Symmetry Detection

The fundamental idea behind this class of propagation functions is to identify symmetries of a fixed (usually small) size. We call such symmetries *sub-symmetries*, as they typically form a small local piece of a larger-scale symmetry. While a sole sub-symmetry might not be considered relevant for the scene's structure due to its limited size, each sub-symmetry detected provides some evidence for the existence of a more relevant large-scale symmetry. Combining the detected sub-symmetries stochastically leads to both reliable and robust information about the scene's structure. What makes this approach especially appealing for

symmetry feature extraction is the fact that one can utilize the detection of *complete* sub-symmetries in order to collect evidence for *partial* large-scale symmetries without explicitly handling the case of missing data.

N-Let Sub-Symmetries

Similar to [Berner et al. 2008], we utilize the metric distance between feature points to identify symmetries. If we can find two pairs of feature points in the scene which have a similar distance, this gives (usually small) evidence that the environments of those point-pairs might be symmetric to each other. When speaking of symmetry features for rigid symmetries, isometric feature points across instances of a symmetry are actually guaranteed (as long as the instances have at least two feature points each). We use the Euclidean metric, which allows to find rigid symmetries. Other metrics can be easily swapped in, for example to enable support for intrinsic symmetries.

Using feature-pair isometries to detect sub-symmetries – although conceptually simple – has two drawbacks. At first it usually is not very distinctive and provokes a lot of false positives. Secondly two feature points do not constrain a rigid transformation, except when adding normal and principal curvature direction information. While adding this information solves the problem of the partially non-constrained transformation, both normal as well as principal curvature directions can be unstable when derived from erroneous input data points. Also normal directions are badly constrained at sharp edges and principal curvature directions are badly constrained at circular peeks. Unconstrained symmetry transformations can be partially handled by our framework as we will see in the part on “Propagating Neighbors”. The bad selectivity however poses a major memory usage complexity problem for explicit feature guess representations. As we will see in the section on result data complexity analysis, we rely on a sufficient selectivity to limit the number of resulting feature guesses.

A straight-forward solution to both problems is to consider not only pairs, but larger tuples of feature points. For 2D manifolds, a well-chosen three-tuple is sufficient for constraining an intrinsic translation. To fully constrain a rigid transformation with respect to the Euclidean metric in 3D space, a 4-tuple is required (without any need to assume a manifold scene).

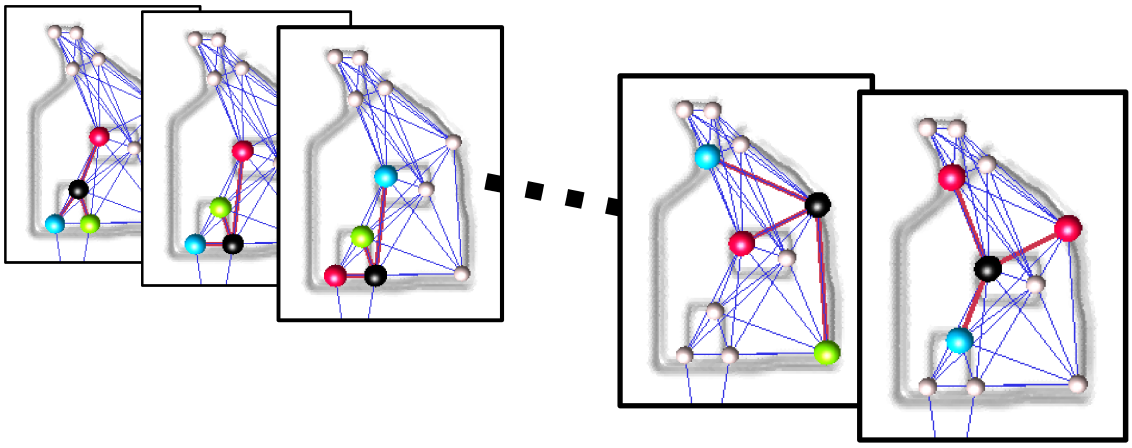

Figure 23: Quadlet examples; In the rightmost case, the red points are ambiguous with respect to their distances to the blue and black point (blue lines are from the neighborhood graph)

Larger tuples of feature points impose a complexity problem unfortunately, as the number of such n-tuples in a scene with $|\Xi|$ feature points is $|\Xi|^n$. We limit the points which constitute an n-tuple to points that are neighbors to each other, meaning that we use locally contiguous n-tuples only. Some points' neighborhood here refers to the k nearest neighbors in Ξ. The (usually realistic) assumption behind this is that sub-symmetry structures can be found at a scale not too much larger than the local average distances between feature points (which is *minStructureSize* at least). We call such a local n-tuple of feature points an "n-let". Figure 23 shows examples of quadlets. After having built all n-lets in a scene, isometric correspondence between n-lets (figure 24) can be established by comparing the $0.5n(n-1)$ pairwise distances between the feature points of which any given n-let consists (compare figure 25).

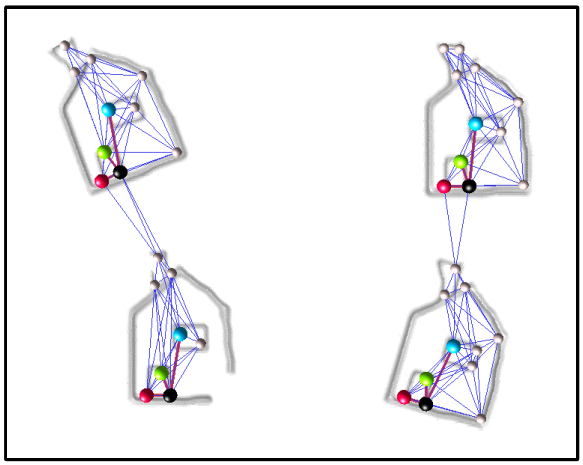

Figure 24: Quadlet correspondence; same colors signal correspondent features in different quadlet instances

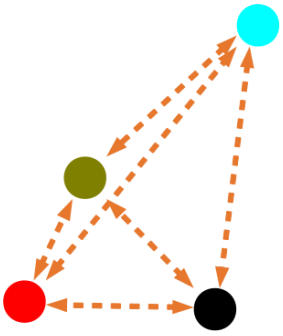

Figure 25: Pairwise distances in a quadlet

Propagating Neighbors

Each n-let $(l_1,\dots,l_n)\in\Omega^n$ provides an implicit local coordinate frame. A point $x\in\mathbb{R}^3$ can be represented by its coordinates $\left(\left\|x-l_1\right\|,\dots,\left\|x-l_n\right\|\right)$. This coordinate mapping is not necessarily injective, depending on whether the rank of the matrix $\begin{pmatrix} l_1^t & \dots & l_n^t \end{pmatrix}$ (which has $l_1,\dots,l_n$ as its columns) is at least four (injective) or not (not injective).

Given a set of corresponding n-lets $L=\{L_1,\dots,L_m\}$, we consider neighborhoods of each instance. The local coordinate transform just introduced allows us to get coordinates for neighbor features which are invariant under rigid transformation, as long as the same transformation has been applied to the n-let which the local coordinate frame gets derived from. By employing this local coordinate transform, it is possible to detect neighbor features which (up to ambiguity resulting from badly constrained n-lets) occur in at least some fraction of the corresponding n-lets' neighborhoods. We will use this information to generate feature guesses especially for those n-let instances, where a corresponding neighbor feature is missing. Let $N_{L_i}:\mathbb{R}^n\to\{0,1\}$ be a function which gives one whenever there is a feature point with given local coordinates (in practice: up to some error) in the neighborhood of the n-let L_i and zero otherwise. The ratio

$$r_L(x)=\frac{1}{|L|}\sum_{i=1}^{n} N_{L_i}(x)$$

for some point $x\in\mathbb{R}^n$ represented in local coordinates then allows us to asses the certainty at which the occurrence of an n-let in L allows to predict the existence of a neighbor feature at the local coordinates x. In other words: having found out that a certain neighbor x occurs in $\sum_{i=1}^{n} N_{L_i}(x)$ out of the $|L|$ corresponding n-let neighborhoods, we conclude that wherever we find a corresponding n-let, there is a point with the same local coordinates as x with a certainty of $r_L(x)$. Therefore, $r_L(x)$ provides an assessment score for how strongly x is "associated" with the n-let symmetry structure represented by L.

The following cases are of special interest:

- The n-let correspondence class L might contain a mixture of n-lets from different large-scale symmetries, which means that these n-lets are not characteristic for a specific large-scale symmetry within the given scene. Such unspecific cases manifest themselves in an accordingly low value of r_L.
- Different permutations of the same n feature points can occur as n-lets in the scene in such a way that they share similar point-to-point distances, and are therefore considered as corresponding to each other. Figure 23 (rightmost picture) contains an example of such n-lets. The problem with these is that the local coordinate frames of different permutations are largely different (in figure 23 : mirrored), while at the same time the permuted n-lets themselves cannot be distinguished from each other. More specifically: there is no canonical form to "prefer" one of the permutations over the others. A single feature point in the neighborhood therefore has multiple local coordinates depending on which permutation is considered. The value of r_L reflects this uncertainty however.

In practice it makes sense to consider only n-let correspondence classes which appear to be relevant in the scene. A straight-forward filtering is to consider classes of a certain minimum size only, where a minimum number of three n-lets per correspondence class is used for all examples.

Now that we have for each n-let correspondence class L a set of ratios $r_L(x)$ for different neighbor feature coordinates $x \in \mathbb{R}^n$, we can use this information to deduce evidence for the existence of a certain feature at local coordinate x in the neighborhoods of all n-lets $L_1, \ldots L_m$. This leads to the generation of a number of feature guesses. In the following paragraphs, we first describe how to restrict the number of feature guesses to relevant ones, by applying different filter criteria. Secondly, we explain how exactly the remaining feature guesses are constructed.

For reasons of memory consumption and processing performance, we only consider neighbors x which have $r_{L_i}(x)$ greater than some constant. All examples use a value of 0.45. The threshold of 0.45 was chosen such that it still permits for example two different large-scale symmetries (with each having similar number of instances) to contain n-lets which match across both symmetries, or alternatively n-lets consisting of points which allow for two matching permutations (e.g. n-lets that allow for mirroring in one axis). Also we require that x occurs in the neighborhoods of at least two n-let instances.

Finally, we also check which $L_i \in L$ qualifies as a target instance for feature propagation. Especially we do not want to propagate neighbor features to instances which have been put into L although not constituting to the same large-scale symmetry which (most of) the other instances in that correspondence class belong to. To detect such outliers among the instances L_i, we calculate for each instance $L_i \in L$ a number p_i. p_i counts the number of neighbors of L_i, which have a matching neighbor in at least $0.45|L|-1$ (for an explanation of 0.45, see above) of the other instances in L. Intuitively, p_i can be thought of as a "participation" counter, saying that the neighborhood of instance L_i "participates" in p_i different neighbor correspondence classes. Some specific instance L_i is considered an outlier, whenever

$$p_i \leq \frac{1}{2m} \sum_{j=1}^{m} p_m$$

In other words, it "participates" in no more than half the number of neighbor feature points that the average instance in L participates in.

We derive one feature guess per sufficiently supported x with respect to L for each non-outlier target instance L_i within any correspondence class $L = \{L_1, \ldots L_m\}$.

Deriving a single feature guess g which proposes the feature at local coordinates $x \in \mathbb{R}^n$ within the neighborhood of an n-let L_i works as follows:

First we derive a certainty γ_g for the guess:

We calculate the total support for the feature point represented in x across all n-let instances by considering the certainties of the underlying feature points $\xi_1, \ldots, \xi_{m'}$ from the neighborhoods of the instances $L_{\sigma(1)}, \ldots, L_{\sigma(m')}$. In this list, $\sigma : \mathbb{N}^{\leq m'} \to \mathbb{N}^{\leq m}$ is an index selection function that selects exactly the n-lets from L, where x exists in the neighborhood. Let $\xi_1, \ldots, \xi_{m'}$ be the concrete feature points with local coordinates x in the different n-let instances' neighborhoods, and let $\gamma_{\xi_1}, \ldots, \gamma_{\xi_{m'}}$ be the associated feature point certainties such that $(\xi_1, \gamma_{\xi_1}), \ldots, (\xi_{m'}, \gamma_{\xi_{m'}}) \in \Xi$. Then the overall support for x is given by

$$s_x = \sum_{j=1}^{m'} \gamma_{\xi_j}$$

We also calculate a rule-of-thumb certainty score γ_{x,L_i} which accounts for the question whether the symmetry characterized by the given n-let correspondence class is valid in the first place. We simplify this score to the question of whether *at least one* of the instances $L_{\sigma(1)}, \ldots, L_{\sigma(m')}$ other than the target instance L_i fulfills the following criteria:

1. Some instance actually exists in the sense that the underlying feature points all exist, which for an n-let instance $(l_1, \ldots l_n)$ is represented by the certainty

   $$\prod_{j=1}^{n} \gamma_{l_j} \text{ with } (l_1, \gamma_{l_1}), \ldots, (l_n, \gamma_{l_n}) \in \Xi$$

2. The same instance also is part of the larger-scale symmetry that x is in. We approximate the certainty for this by a constant $\gamma_m = 0.6$.

Combining both criteria and incorporating the fact that a single instance other than L_i which matches the criteria is sufficient, we get the overall formula

$$\gamma_{x,L_i} = 1 - \prod_{(l_1, \ldots, l_n) \in \left| L_{\sigma(1)}, \ldots, L_{\sigma(m')} \right| \setminus |L_i|} \left(1 - \gamma_m \prod_{j=1}^{n} \gamma_{l_j} \right)$$

The overall certainty for g is then provided by the term

$$\gamma_g = c\, \gamma_{x,L_i} \frac{s_x}{|L|}$$

with c being a general certainty correction factor which is specific to this propagation function. This factor approximates the probability that some feature point ξ is an actually desired symmetry feature point under the condition that ξ gets predicted by the specific propagation function with maximal achievable certainty (i.e. with $\gamma_{x,L_i} s_x |L|^{-1} = 1$ in our case). A value of around $c = 0.07$ has turned out to give robust feature results. Please note that c only limits the maximum certainty that a single first-order guess of the specific propagation function can have. During subsequent feature guess merging, a multitude of such low-certainty guesses can still be merged into a single guess of high certainty.

The distribution τ_g that we use for g is a multiplicative distribution with multiple Gaussian radius sphere distributions as sub-distributions. For each l_j in $(l_1, \ldots, l_n) = L_i$ we create a Gaussian radius sphere sub-distribution with radius x_j, where x_j is the jth coordinate of x. The rationale behind using a multiplicative distribution of Gaussian radius sphere sub-distributions is the fact that the proposed feature point has to match *all* of the coordinates, where each coordinate gives a radius around the corresponding n-let point up to some assumingly normal distributed deviation.

While alternatively a well constrained n-let with $n \geq 4$ or $n \geq 3$ for the general and intrinsic case respectively allows to derive a rigid / intrinsic mapping between corresponding n-lets and one could use this mapping to transform one instance's neighborhood to another directly, the approach of using a multiplicative distribution over Gaussian Radius Sphere distributions has two major advantages:

1. It gives a good representation of the uncertainty in the guessed feature's position, without a need to explicitly assess the degree of constraint for each spatial direction.
2. Even in degenerate cases, where the n-lets do not impose a reliable local coordinate frame (e.g. all points being located on a line), the multiplicative sphere approach does still allow to utilize the limited information that is there (with associated problems for feature guess merging however). Deriving a rigid transformation from such n-lets is either impossible or unstable and largely arbitrary for noisy positions of the n-let's points. For example with all points being approximately located on a line, a minor variation in one feature's positions can cause the local coordinate frame used to derive the transformation to “switch over” completely.

Additionally, we assign a correspondence set to the guess g:

$$K_g = \{(\xi_1, \gamma_g), \ldots, (\xi_{m'}, \gamma_g)\}$$

signaling that all feature points which provided evidence for the feature guess are correspondent to the guessed feature point with a certainty proportional to γ_g.

Implementation Details

We use a default value of $n=4$. N-lets are generated by iterating over each feature point $(\xi, \gamma_\xi) \in \Xi$ and for each feature ξ building all n-tuples with a set of $k_{\text{n-let}}$ neighbors. The $k_{\text{n-let}}$ neighbors are selected such that each two of them has a minimum distance of $8\sqrt{2\sigma^2}$. The Set of neighbors is derived by iterating over the features in order of ascending distance to ξ. For each such candidate feature, the minimum distance requirement with respect to all neighbors found so far is checked and if met, the candidate is added to the set of neighbor features. The minimum distance requirement is there to ensure that the distances between points constituting to an n-let are significantly larger than the standard deviation for point-to-point distances $\sqrt{2\sigma^2}$, which helps to establish a stable local coordinate frame based on the resulting set of n-lets. For all examples shown, it is $k_{\text{n-let}} = max\{6, n+1\}$. As the number of generated n-lets is

$$\frac{k_{\text{n-let}}!}{(k_{\text{n-let}}-n)!},$$

$k_{\text{n-let}}$ must not be too large if memory consumption and runtime are of concern.

While generating n-lets over the set of feature points Ξ, we employ an $0.5n(n-1)$ dimensional KD-tree $K_{\text{n-let}}$ to check whether a newly generated n-let can be considered as corresponding to an n-let that has been generated before. To identify corresponding n-lets in $K_{\text{n-let}}$, we use the pairwise distance vector for each n-let as described above.

If $K_{\text{n-let}}$ contains an n-let with a pairwise distance vector similar to the one of the currently generated one, we merely store the tuple of n feature points of the new n-let into the existing n-let data structure as an additional instance. Otherwise we insert the new n-let into $K_{\text{n-let}}$. For deciding whether merging with an existing n-let is possible, we use a matching tolerance of $2\sqrt{4\sigma^2}$ for each component of the distance vector independently. Here the factor 4 is a trade-off between false positive and false negative merging assumptions and the variance of $4\sigma^2$ results from the fact that we are considering the difference between two distances, which themselves are differences between two independent normally distributed point positions of variance σ^2 each. If multiple matches exist, merging is performed with the one that is closest to the newly generated n-let with respect to the two-norm distance between both n-let's distance vectors. An additional merging candidate validation using the two-norm (in contrast to thresholding each pairwise distance independently) can optionally be performed in order to make merging candidate selection invariant towards the direction of the difference between the distance vectors.

A similar approach is taken to find corresponding neighbor features once local coordinate frames have been established in the form of an n-let correspondence class L. While iterating over all instances $L_i = (l_1, \ldots, l_n)$, each of the k_{prop} nearest feature points around l_1 (with $k_{\text{prop}} = 12$ for all examples) are characterized by their local coordinates with respect to L_i. For each such neighbor feature, a KD-tree K_L (which is individual for each n-let correspondence class L) is checked for whether it already contains a point with similar local coordinates from another instance. A per-coordinate tolerance of $1.5\sqrt{4\sigma^2}$ is used here.

Gaussian radius sphere sub-distributions are generated with a variance of $3\sigma^2$, which accounts:

1. the double-variance in the measured distance between the corresponding reference point in the source n-let and the source feature point being propagated
2. plus the uncertainty in the position of the center point of the Gaussian radius sphere, which is the corresponding reference point in the target n-let.

Comparable to the current implementation of feature guess merging, establishing correspondence between both n-lets and neighbor features has the issue of depending on the actual order in which n-lets and neighbors respectively are inserted. Please refer to chapter 4.5.2 for a more detailed discussion of this issue.

Result Data Complexity Analysis

As mentioned in chapter 4.5.2, the number of generated guesses is crucial in order to not exceed limits on memory consumption, as well as for runtime complexity. Thanks to the way in which only sufficiently supported neighbor features of a class of corresponding n-lets lead to feature guesses, the number of guesses generated by the n-let-based propagation functions can be shown to be no more than linear in the number of feature points.

We generate a total number of

$$\frac{k_{\text{n-let}}!}{(k_{\text{n-let}}-n)!}$$

n-lets per feature point $(\xi, \gamma_\xi) \in \Xi$, with $k_{\text{n-let}}$ being the (constant) number of neighbors considered in n-let generation. Therefore we have a total number of n-lets

$$N = \frac{|\Xi| k_{\text{n-let}}!}{(k_{\text{n-let}}-n)!} \in \Theta(|\Xi|)$$

Due to the merging scheme employed, each n-let can participate in one correspondence class only, as it is *either* stored into the KD-tree as a new n-let *or* added to an existing n-let data structure as an additional correspondent instance.

Each n-let has exactly k_{prop} neighbor features which are considered for feature guess generation, which again is a fixed constant. Within some n-let correspondence class L, each of the $k_{\text{prop}}|L|$ neighbors can only be assigned to a single neighbor feature correspondence class, as the same argument as for n-let correspondences applies.

After clustering neighbor features into correspondence classes, we throw away correspondence classes which contain less than $0.45|L|$ neighbors. Therefore, the maximal number of neighbor correspondence classes that remain for L can be no more than $k_{\text{prop}} 0.45^{-1}$, as each correspondence class "occupies" at least $0.45|L|$ of the $k_{\text{prop}}|L|$ neighbor features.

The total number of feature guesses generated for L can now be constrained. We generate one feature guess for each n-let in L per remaining neighbor correspondence class at maximum. We get for the number of feature guesses

$$|G_L| \leq \frac{k_{\text{prop}}}{0.45}|L| \Rightarrow |G_L| \in O(|L|)$$

Even more, this constrains the number of feature guesses generated for the whole scene. As the sum of all n-let correspondence class sizes $|L|$ can be no more than the number of n-lets N, we generate no more than

$$\frac{k_{\text{prop}}}{0.45} N = |\Xi| \frac{k_{\text{prop}} k_{\text{n-let}}!}{0.45(k_{\text{n-let}} - n)!} \in O(|\Xi|)$$

feature guesses.

There is one remaining issue about the result data memory consumption however. Although the number of feature guesses is no more than linear in the number of feature points, the size of feature guesses g is not constant, as each contains correspondence information K_g which can be linear in the number of feature points. A trick can however be used to limit memory consumption: Instead of storing pairs (ξ_i, γ_g) to denote a correspondence with ξ_i at certainty γ_g, we can instead store relative certainties. More exactly, we store pairs $(\xi_i, 1)$ in which the relative certainty 1 has to be multiplied by γ_g before yielding the actual correspondence certainty between the feature point proposed by g and ξ_i. While the guess certainty γ_g can be different for each generated guess, the set K_g' of correspondence pairs with relative correspondence certainties is the same across all guesses $g \in G_L$ generated for some n-let correspondence class L. Furthermore, each such correspondence set K_g' can have no more than $|L|$ elements, as that is the maximum size of any neighbor feature correspondence class for L. This leads to a limit on the number of *different* sets K_g' of $k_{\text{prop}} 0.45^{-1}$, each with no more than $|L|$ correspondence elements for each n-let correspondence class L. Storing each such set only once and using a constant-size pointer in the generated feature guesses to point to the associated correspondence set restricts the memory consumption to $O\left(k_{\text{prop}} 0.45^{-1} |L|\right)$ for each L, and

$$O\left(\frac{k_{\text{prop}}}{0.45}|N|\right) = O(|N|) = O(|\Xi|)$$

in total.

5 Evaluation

We evaluate the behavior of our framework and of the incorporated sub-symmetry detection on one artificial and a number of scanned real-world data sets.

The following data sets are used:

– Embossed houses (figure 26): We generated this artificial example by drawing the house structure, applying a small-scale Gauss filter, copying, translating and rotating the house drawing to four different locations, adding a few small mistakes to one of the instances (including a missing edge) and then embossing the image into a point sampled 3D plane. The data set was sampled to 37797 points. Please note that the white area in figure 26 also contains points.

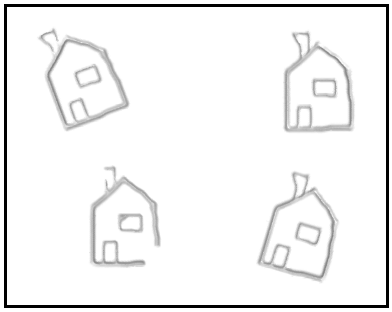

Figure 26: Embossed houses data set

– Old Hannover town hall (figure 27): This scan of a historic building in Hannover was done using 3D laser scanners. For our experiments, we cut out a part of the scene. Limited outlier point removal was performed and the whole data resampled to 82818 points. (courtesy of the Institute for Cartography and Geoinformatics Hannover).

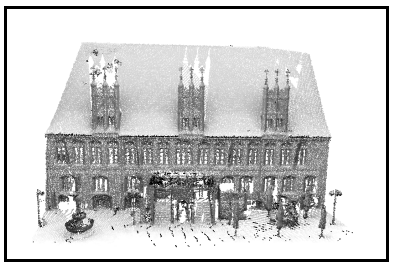

Figure 27: Hannover town hall data set

– Zwinger (figure 28): The Dresden Zwinger data set is another building digitalized by the use of 3D laser scanners. We sampled the scene to a total of 76231 points. (courtesy of M. Wacker)

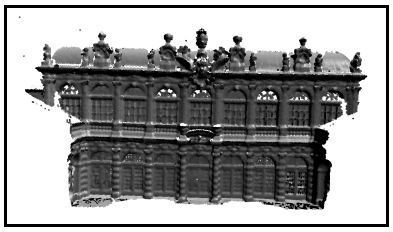

Figure 28: Zwinger data set

– Ambasciata del Brasile (figure 29): This digitalizations of the front of the Brazil embassy in Rome was obtained by 3D data reconstruction from photographies, and has a comparably high noise level. The scene was sampled to 455938 points. We applied limited outlier points removal. (courtesy of Blizzard Entertainment Inc.)

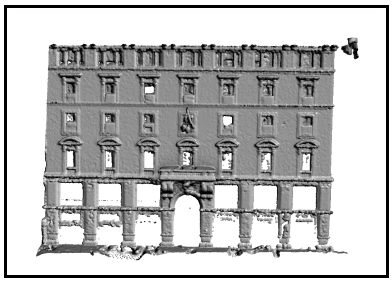

Figure 29: Ambasciata del Brasile data set

– Thai elephant statue (figure 30): The Thai statue scene was obtained using a 3D laser scanner. We use only an excerpt from the statue, consisting of 144392 points. (from the Stanford 3D Scanning Repository)

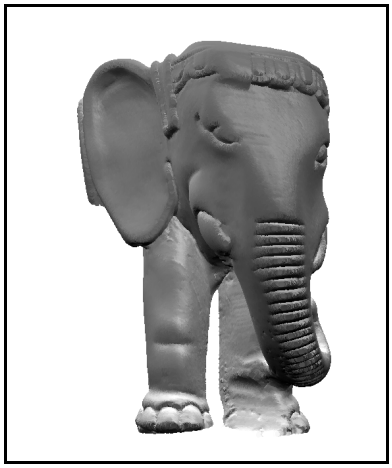

Figure 30: Thai statue data set

5.1 Feature Extraction Results

Figures 31 to 34 show the results of feature point extraction for different data sets. Red points are initial features, in other words those which have been extracted based on local interest purely, while blue points are those features that where selected only with the help of feature propagation.

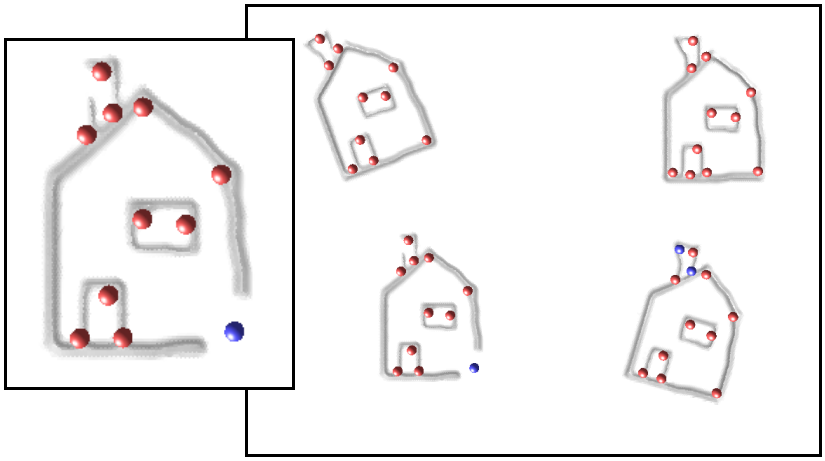

Figure 31: Embossed houses feature points

The Embossed houses results were achieved after two iterations (one iteration for the selection of interest features, second with propagation) with slight tweaking of $minStructureSize$ and σ^2 parameters, slippage interest was used. Processing of the houses scene took 2 minutes and 1 second. While the chimney features are not fully consistent across all instances, the completion of the feature point in the bottom right corner of the house is remarkable here, as purely local criteria would not have allowed to detect this feature point in the incomplete instance. Also interesting is the added feature in the top left of the chimney of the bottom right house instance. No other instance has a feature at the corresponding location, so this feature completion must have been due to an erroneous sub-symmetry match, which together with the local interest raised the combined score just enough to qualify that point for feature selection.

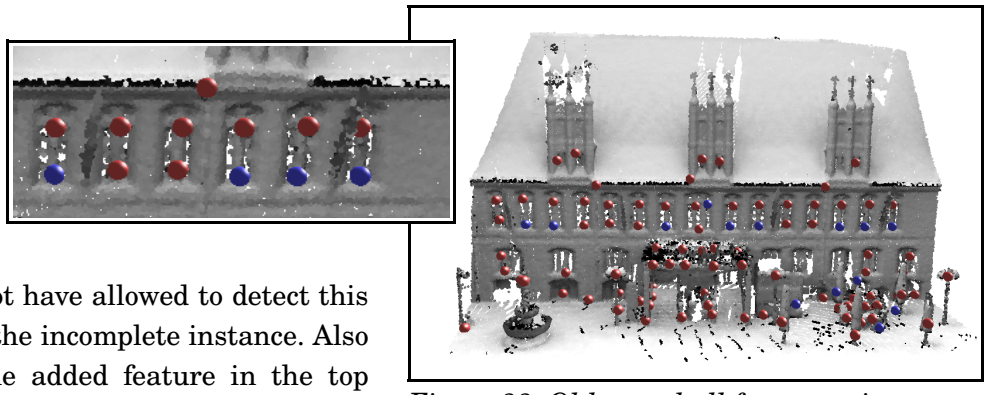

Figure 33: Old town hall feature points

The old Hannover town hall data set shows well the selective completion of features in the upper row of windows. As a consequence of noise and holes in the data, eight of those feature points were absent in the initial interest-based feature selection and could be completely recovered by feature propagation. We performed seven iterations for this data set, using the multiplicative interest combiner function and slippage-based interest. The overall running time of our algorithm was 2 minutes and 58 seconds.

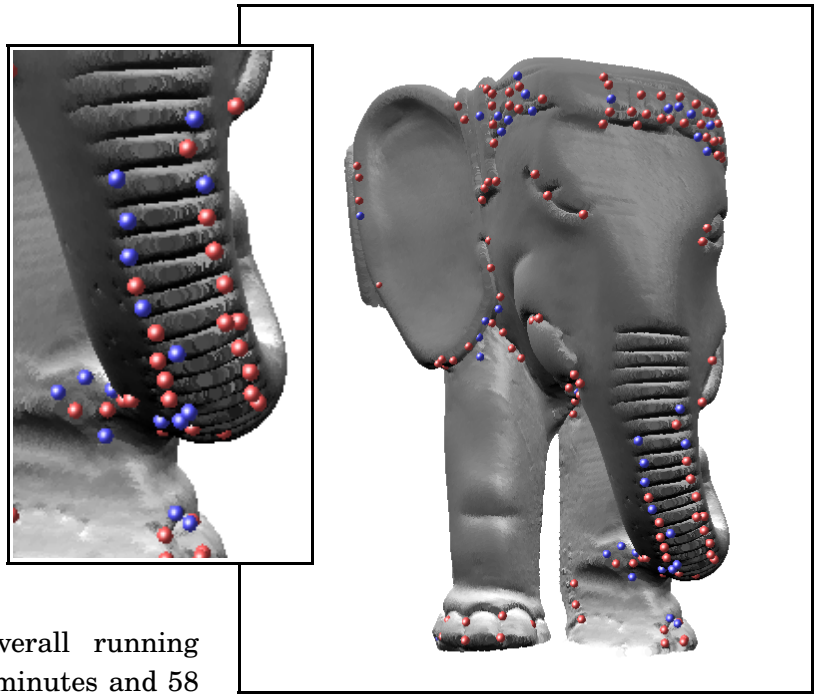

Figure 32: Thai elephant feature points

For the elephant from the Thai statue we used slippage-based interest and the default combiner without mandatory local interest. Feature points on the elephant's trunk were completed up to some degree, with seven correct ones being added there due to feature propagation. Feature extraction took 4 minutes and 29 seconds at seven iterations.

The Dresden Zwinger data set took 5 minutes and 14 seconds to get processed. The multiplicative interest combiner was deployed here with slippage-based interest scores. We ran seven iterations.

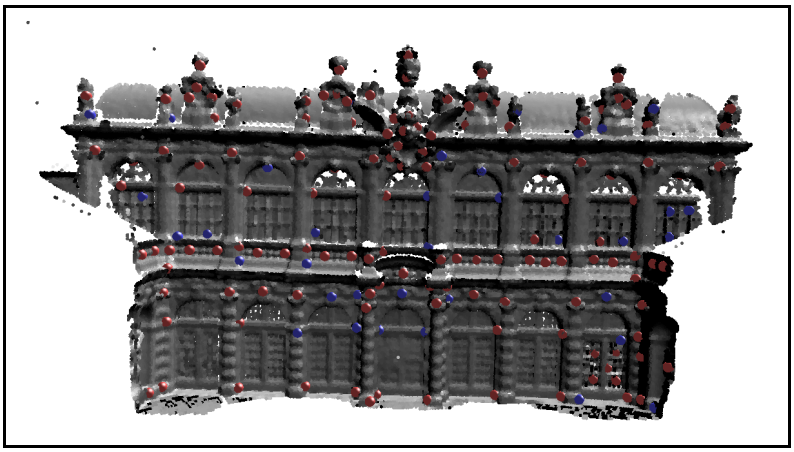

Figure 34: Zwinger feature points

All running time measurements mentioned here do not include the time required for slippage and/or curvature analysis. They do however include a run of our propagation function applicability analysis algorithm from chapter 4.4. Measurements were performed on a Intel Core i7 620M mobile CPU with two cores and four gigabytes of main memory.

5.1.1 Choice of Parameters

We evaluate the effects of different parameters and interest function constellations with respect to feature extraction. Specifically, we compare the initial feature points provided by slippage- and curvature-based interest and also consider the combination of both. With

respect to interest-based feature point selection, we also examine different feature point selection thresholds. Finally, we compare feature extraction results after a varying number of iterations of our framework.

Interest Features

We have experimented with slippage and Gaussian curvature-based interest functions as well as combinations of both. Figure 35 compares initial interest-based feature points that were selected based on slippage, curvature and combined interest respectively. Curvature-based feature selection reveals many unexpected feature points, and at the same time lacks many others. Overall, feature point placement is comparably irregular with curvature-based interest. Slippage features on the other hand are much cleaner and more regular, although a few of the useful features that were found in the case of curvature interest are missing here. A combination of curvature and slippage interest with weights of 0.9 for slippage and 0.6 for curvature interest yields the most complete set of feature points with only few false positives (these parameters were used in figure 35).

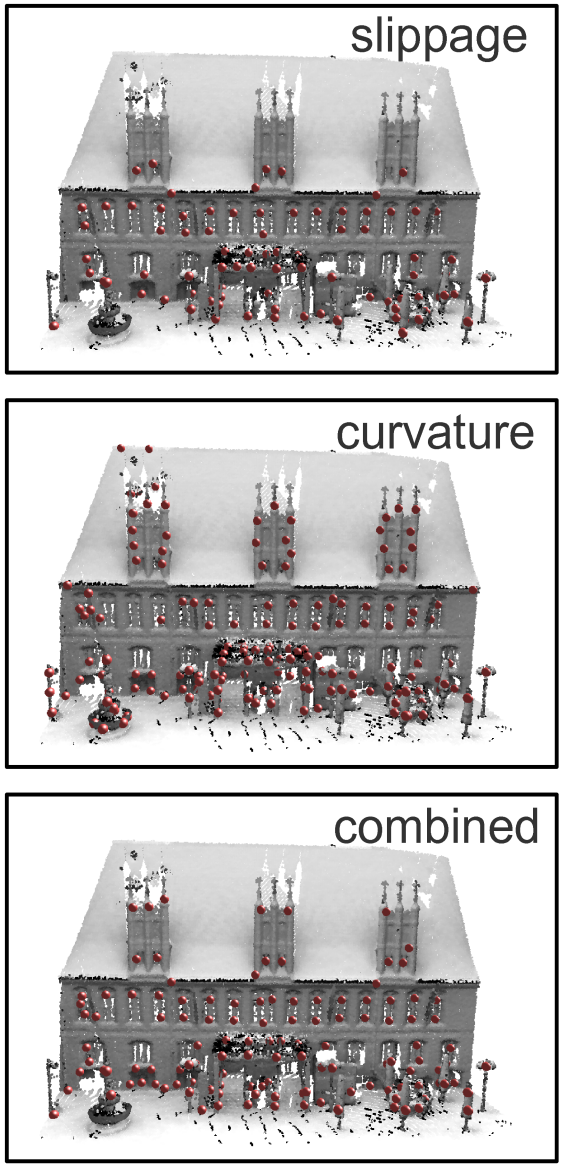

Figure 35: Comparison of interest functions

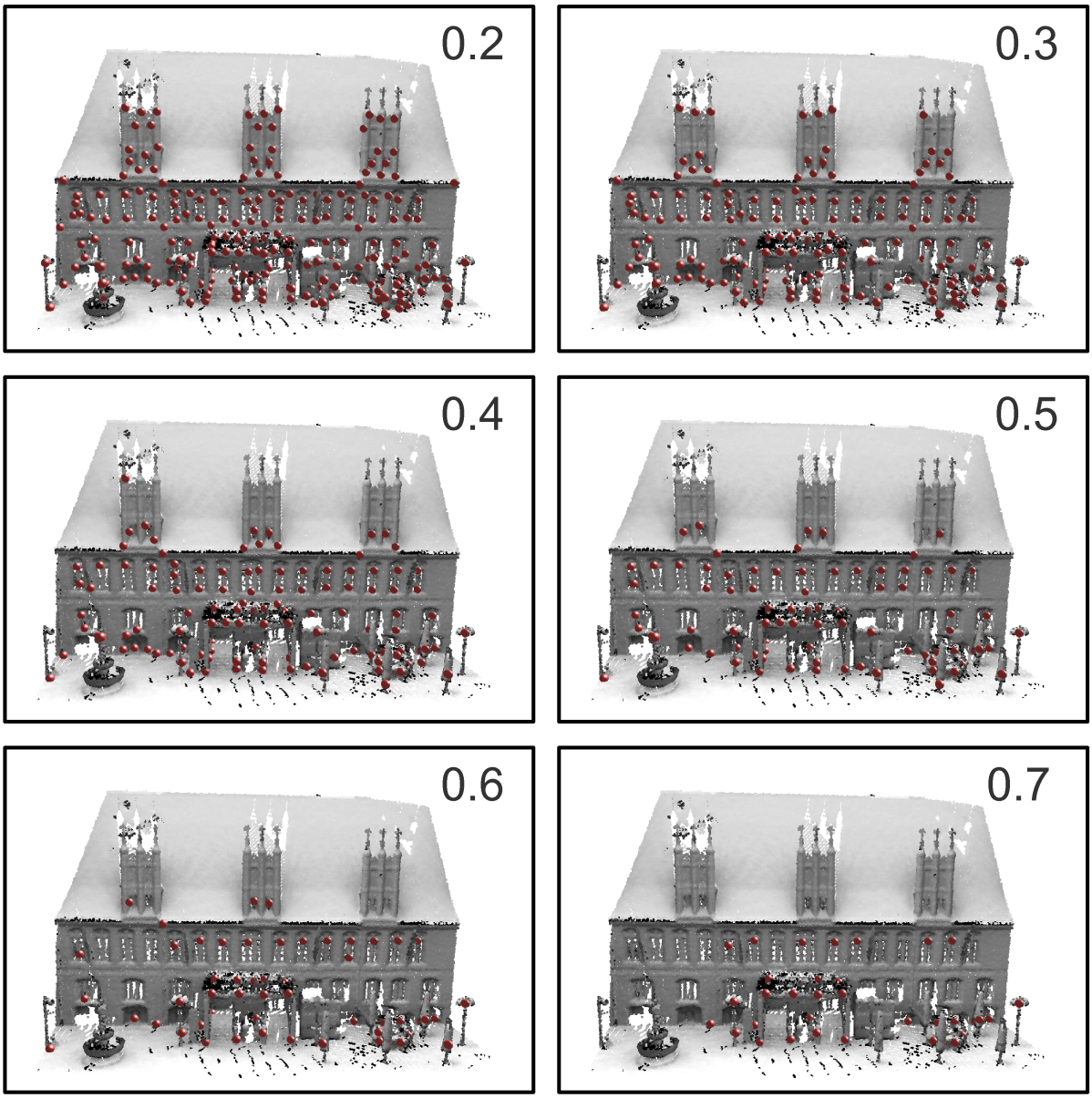

Figure 36: Comparison of selection thresholds

Our selection function implementation uses a threshold parameter to select only feature points with sufficient combined evidence. Figure 36 compares the selected features for different selection thresholds, based on interest from the slippage interest function. We use a threshold of 0.5 for all other examples, as it gives a relatively stable feature selection while still containing enough feature points to get sub-symmetry-based feature propagation started. When comparing the lower-threshold interest-based feature selection results to the feature set that we achieve for the old town hall data set after multiple iterations of our framework in figure 33, it is clearly visible that our algorithm selectively recovers symmetric feature points, namely the ones in the upper row of windows. In other words, the resulting feature points are much closer to a set of perfect symmetry features for this data set than what is achievable by purely local feature selection.

Number of Iterations

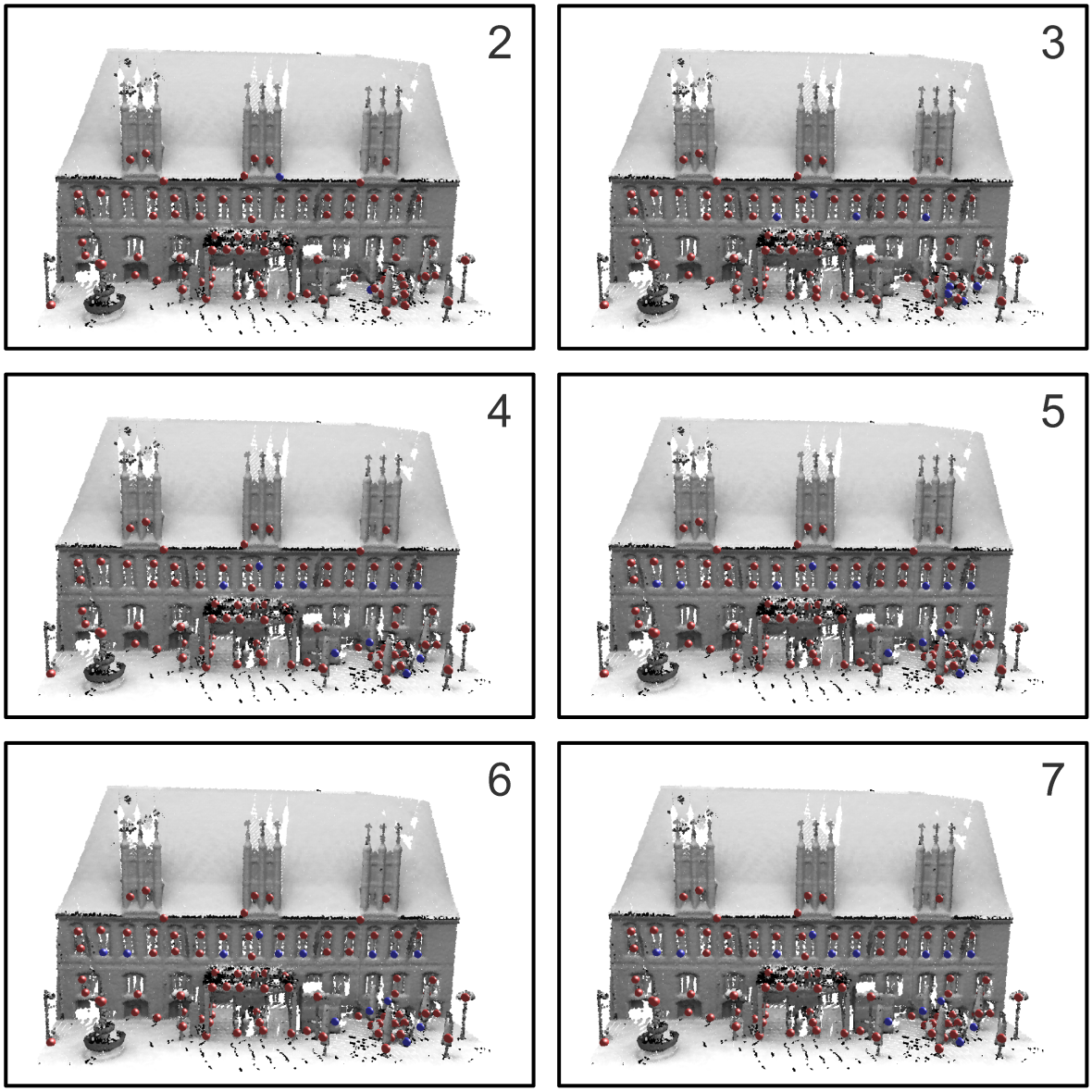

Figure 37: Feature point set after different numbers of iterations for the old town hall data set; Blue feature points were added due to feature propagation

The exact number of iterations required for a stable result differs between scenes. Figure 37 shows feature points after a different number of iterations (multiplicative interest combiner, slippage interest), where iteration one is equivalent to the interest-based slippage features shown in figure 35 and not repeated here. For this data set, stabilization of the feature point set occurs after the 5th iteration. For the embossed houses on the other hand, even two iterations are sufficient. We use seven iterations for most examples to leave enough head-room for the feature point sets to converge.

5.2 Correspondence Extraction Results

One example for feature correspondence extraction was already given in figure 12 for the embossed houses data set.

Figure 38 shows the set of extracted correspondences for the town hall data set (multiplicative interest combiner, slippage interest, seven iterations). Two main feature correspondence classes were detected, namely the upper row of windows and features from a small tree in front of the building.

Figure 38: Correspondence classes for Hannover town hall data set; Large spheres denote corresponding features, one color (including white) per correspondence class; small gray spheres: unassigned features

Figure 39 shows detected correspondence classes for the Thai elephant dataset (slippage interest, seven iterations). Correspondence between feature points on the trunk was nicely established, while other correspondences in the scene where not detected with sufficient certainty.

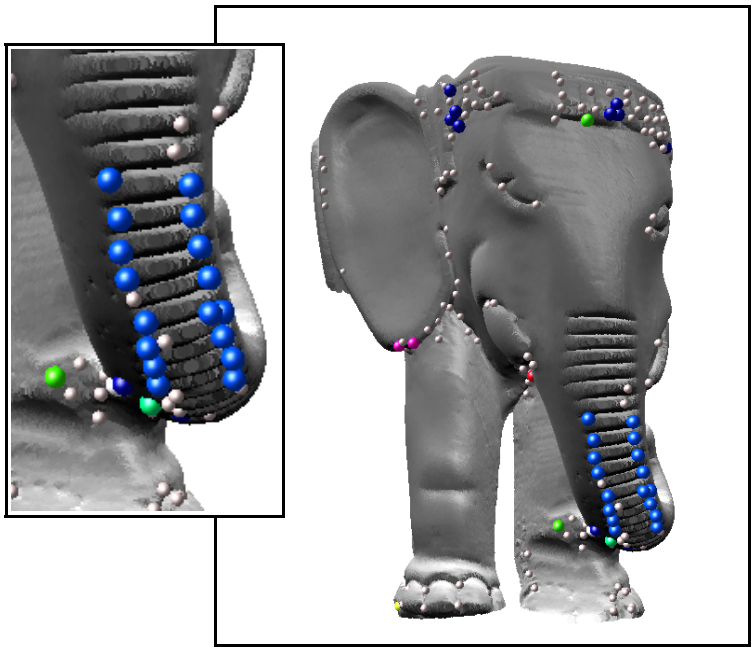

Figure 39: Correspondence classes for Thai elephant; Large spheres denote corresponding features, one color per correspondence class (excl. white); small gray spheres: unassigned features

Figure 40 shows detected correspondence classes for the Zwinger data set (multiplicative interest combiner, slippage interest, seven iterations). While a total of four features were mis-assigned (figure 41), many useful correspondence classes could be established.

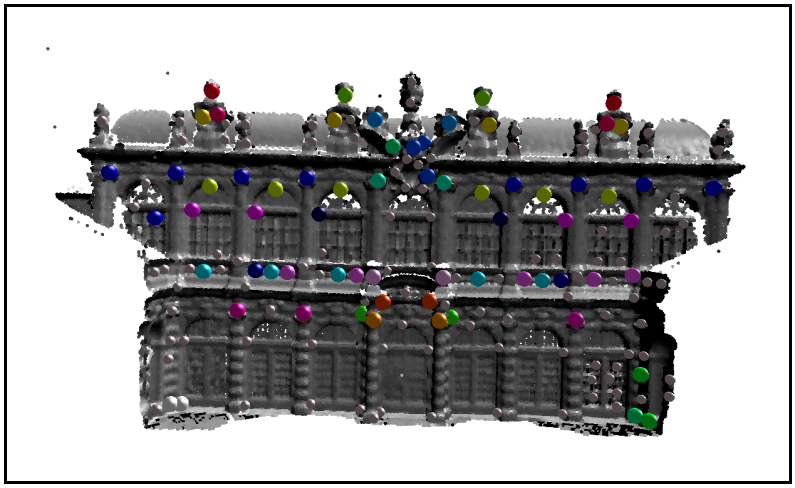

Figure 40: Zwinger correspondence classes; Large spheres denote corresponding features, one color (including white) per correspondence class; small gray spheres: unassigned features

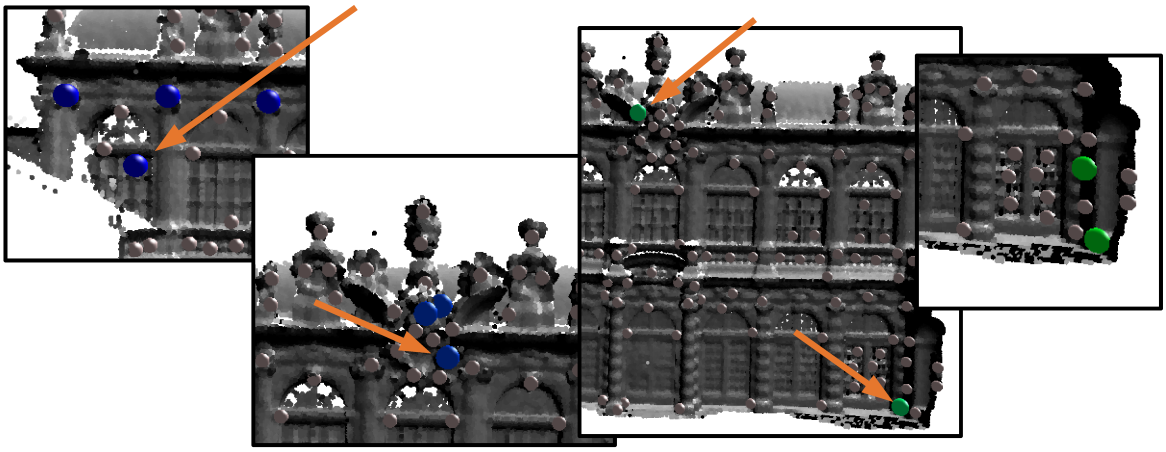

Figure 41: All of the four false correspondences detected in Zwinger data set

5.2.1 Choice of Parameters

In chapter 4.7 we introduced a certainty threshold parameter of usually 0.3, telling when two feature points should be considered correspondent.

Figure 42 illustrates results of different parameter choices for the old Hannover town hall data set (multiplicative interest combiner, slippage interest, seven iterations). As we can see, a threshold value of 0.3 is the lowest one without false-positive correspondences and also the highest one, which still gets all of the upper window-row features as correspondent. For other scenes the optimal choice of this parameter can differ (we used 0.3 for all examples though).

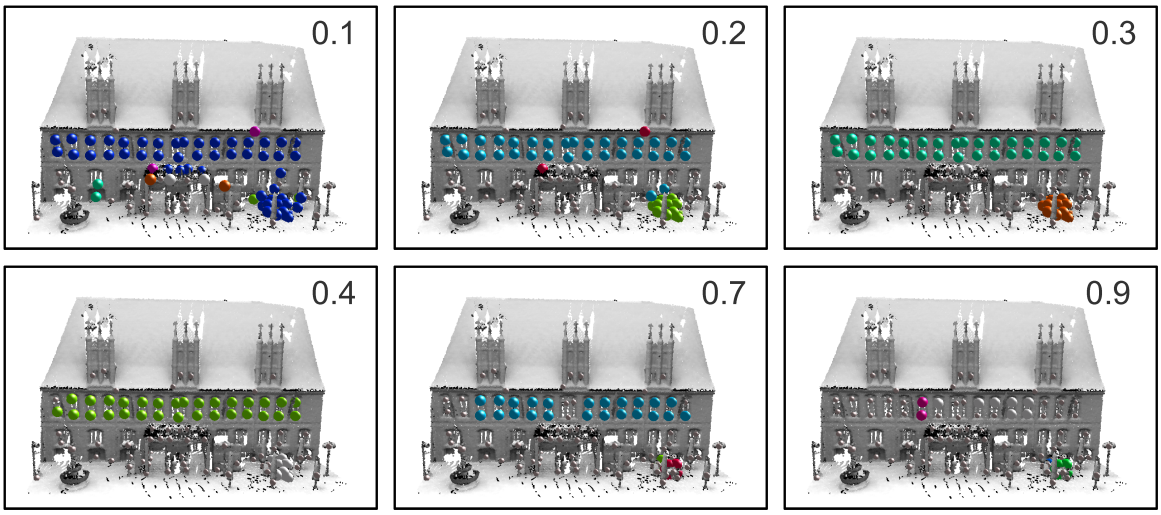

Figure 42: Comparison of different correspondence threshold parameters; Large spheres denote corresponding features, one color (including white) per correspondence class; correspondence class colors are random for each iteration independently; small gray spheres: unassigned features

5.3 Propagation Function Applicability

We tested different values for n in the n-let sub-symmetry propagation function and checked the resulting applicability scores as calculated by our propagation function applicability analysis algorithm (compare chapter 4.4). For comparison, we also implemented a propagation function which just creates a number of 100 feature guesses at random positions in the scene. The tests were run using the old Hannover town hall data set.

Figure 43 compares the feature points that got selected with the different propagation functions. The random propagation function leads to three correct and four incorrect feature point completions. The comparably high ratio of correct feature points is due to the use of the multiplicative interest combiner, which limits the negative impact of misplaced feature guesses somewhat. For $n=3$, no false positives exist but only two of the missing features in the upper row of windows got completed. $n=4$ leads to the completion of all missing features in these windows. $n=5$ and $n=6$ both were not able to recover any of the features in the upper row of windows. Two different effects can be observed here when changing the value for n in the n-let propagation function:

1. For low values of n, the pairwise distances of n n-let are not sufficiently discriminative. This leads to many false positives in the generated n-let correspondence classes. As a consequence, it gets harder to find neighbor features that are consistent across a sufficiently large fraction of the corresponding n-let instances. However, we generate feature guesses only for sufficiently consistent neighbor features (for details and the specific threshold used: see chapter 4.8.4).

   This effect is especially strong for $n=2$, where not a single feature guess gets generated for the old Hannover town hall data set.

2. When the value of n gets higher, an increasing number of feature points have to match in order to consider any two n-lets as corresponding. Depending on the noise level of the feature point set used for n-let generation and the number of missing features in that set, finding a sufficient number of matching n-lets can be difficult.

   This effect is especially strong for $n=5$ and $n=6$.

Please note that the dramatic difference between for example $n=4$ and $n=5$ is also due to the iterative nature of our framework. In figure 37, we saw that for $n=4$, the first new features get introduced with the third iteration (with the second one only optimizing the positions of feature points). Three of the missing feature points from the upper row of windows are recovered in that iteration, which enables the algorithm to complete the rest of the missing features in the subsequent iterations. With $n=5$ on the other hand, this process does not get started in the first place. In other words: a propagation function which leads to only slightly worse results in an early iteration, can result in massively worse results after a larger number of iterations.

The following applicability scores were derived (running times are for seven iterations):

random	0.126	
n = 2	N/A (no feature guess was generated)	
n = 3	0.478	running time: 41s
n = 4	0.565	running time: 2m 58s
n = 5	0.622	running time: 2m 13s
n = 6	0.611	running time: 4m 48s

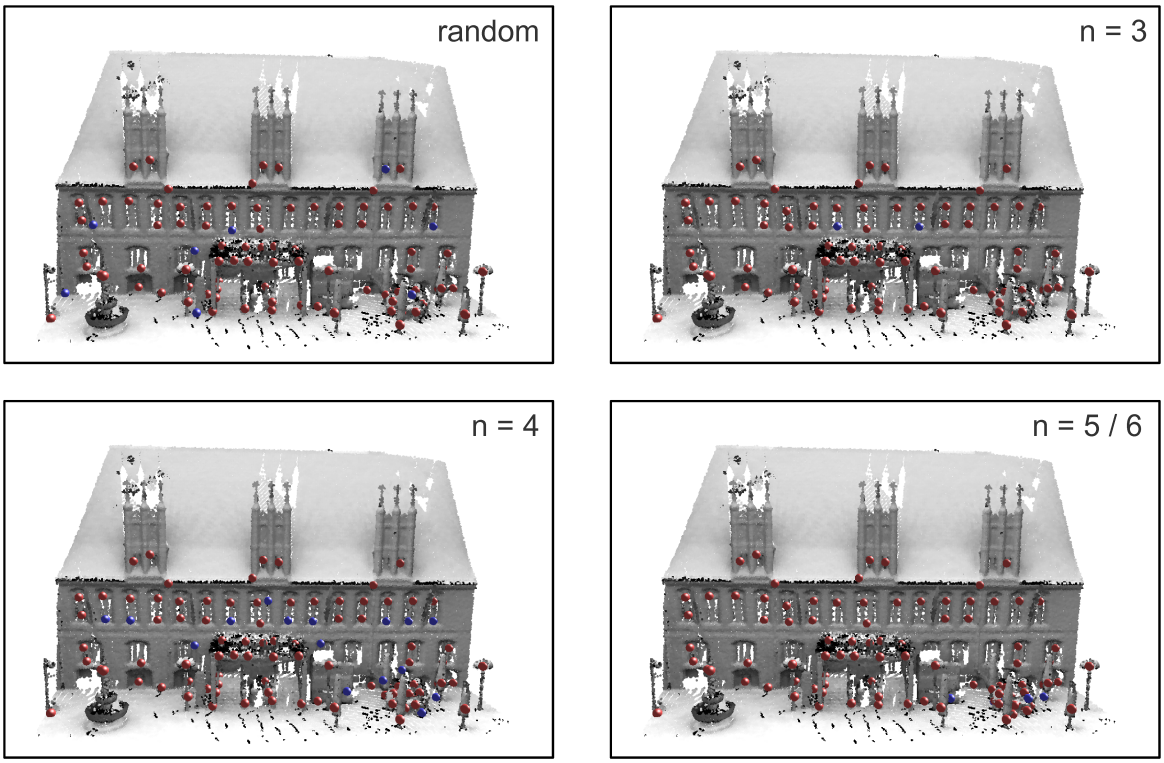

Figure 43: Comparison of random and n-let propagation functions

While the scores approximately reflect the increasing correctness of propagations and especially applicability analysis was able to detect the low applicability of the random propagation function, the resulting scores do not strictly correspond with the quality of feature selection results. As we see in figure 43, usable feature guesses were generated only for $3 \leq n \leq 4$, with $n=4$ clearly giving the best results. The best applicability score however is reached with $n=5$. The reason for this inconsistency between the applicability scores and resulting feature points can be explained by the fact that our applicability score rewards propagation functions which stick closely to the existing interest-based feature points and predict little new.

5.4 Problems and Limitations

While using global criteria for feature extraction is crucial for the extraction of symmetry features, it also has the side effect that the local result quality can be influenced by global data in a negative way. This is especially true with our n-let propagation function, as we only propagate points that are at least partially consistent across instances of an n-let correspondence class (compare the section on "Propagating Neighbors" in chapter 4.8.4). After adding additional data to a scene, n-lets which were sufficiently discriminative to have consistent neighbor features in the original scene might become insignificant with respect to their neighbors in the extended scene. How severe this problem is, strongly depends on the scene and the additional data added. For the old Hannover town hall data set, our framework was not able to extract a feature set comparable to what we have seen in figure 33 anymore, after adding some additional structure (including a large tree and side walls), when sticking to default parameters (figure 44). This problem can be worked around by pre-segmenting the scene into large-scale building blocks and applying our framework to each part independently.

Figure 44: Features for Hannover town hall data set with added scenery and side walls

The n-let sub-symmetry propagation function requires at least n feature points to be consistent across instances of some symmetry, in order to get a propagation process started. If the initial features are very unstable, we cannot establish sub-symmetry correspondences which we would need to recover from such situations. The Ambasciata del Brasile data set for example does not give stable slippage nor Gaussian

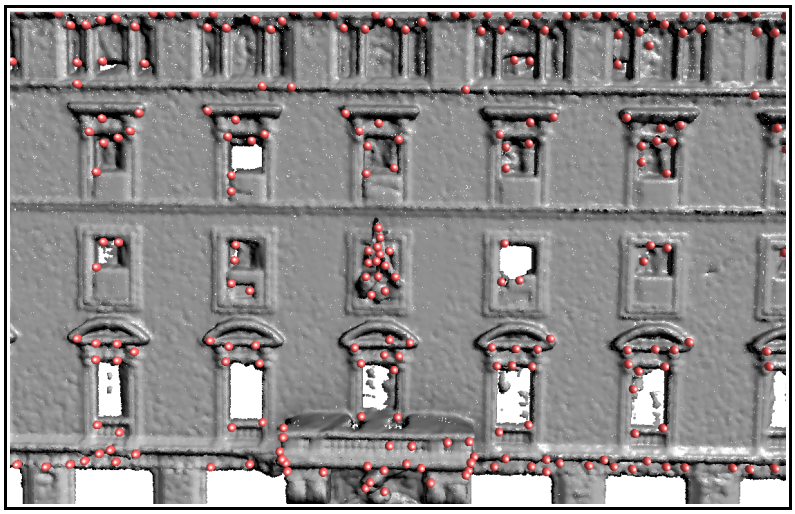

Figure 45: Initial features for the Ambasciata del Brasile

curvature interest scores, due to the high level of noise in the data. Figure 45 shows an excerpt of the initial feature point set we got with slippage-based interest. Our algorithm was not able to improve the feature point set for this scene.

While we did not observe it with the tested data sets and our current implementation, there is the potential problem of divergent behavior. While iterating, more and more erroneous feature points might potentially get added and those very points can then lead to false feature guesses which in turn make even more points getting selected as features in the next iteration. Using the multiplicative interest combiner function might reduce this risk, as feature points can only be selected at places which had at least some local evidence in the scene initially.

6 Conclusion

Using our framework and the idea of sub-symmetry detection, we were able to significantly improve the quality of point features for most input data sets. Specifically, feature points missing in the initial local-criteria-based feature point selection got selectively completed for multi-instance symmetries in the respective scenes. Of course, the resulting feature sets are not a perfect set of symmetry features with regard to all existing symmetries in the scenes. We however approached the goal of extracting sets of symmetry features much closer than the local-criteria-based feature point extraction methods we have examined (namely slippage- and Gaussian curvature-based approaches). We also were able to demonstrate the robust performance of sub-symmetry detection with regard to feature-to-feature correspondence analysis.

While some problems remain (compare chapter 5.4) and our method is not suited well for every input scene in its current implementation, the framework we have proposed leaves options for further improvement. Many additional instances of interest, combiner, selection and especially propagation functions are yet to be considered. While we have proposed one or two concrete implementations for each function type, for example a DoG-based interest function might improve feature results for some scenes. Even more so, virtually any symmetry detection algorithm can be adapted to operate within a propagation function, with correspondingly different classes of detectable symmetries.

Our sub-symmetry detection approach should be elaborated in more detail in the future. Considering local feature descriptors in addition to the metric relation of feature points to each other could lead to more robust results with some scenes. A potential descriptor should be selected under consideration of robustness with respect to noisy and/or incomplete input point data. Changing the currently used euclidean metric to for example an intrinsic or even more sophisticated metric in the detection of sub-symmetries, poses an interesting topic for future investigation. Changing the metric should allow to extract symmetry features with respect to deformable symmetries with only minimal changes to the algorithms involved.

For simplicity reasons, we have focused on point features. The general framework however should be adaptable to other classes of features as well. How to perform such adaptation and what performance results from those might be another subject of future investigations.

Finally, a deeper mathematical analysis of our framework might help to better understand and describe its behavior in an exact way. It might even be possible to derive a continuous variant of our symmetry feature extractor, presumably by using tools from continuous stochastics and analysis.

7 References

[Alt et al. 1988] Alt, H., Mehlhorn, K., Wagener, H., Welzl, E.:
Congruence, Similarity, and Symmetries of Geometric Objects
In: Discrete & Computational Geometry 3, 237-256, 1988

[Berner et al. 2008] Berner, A., Bokeloh, A., Wand, M., Schilling, A., Seidel, H.-P.:
A graph-based approach to symmetry detection
IEEE/EG, International Symposium on Volume and Point-Based Graphics, 2008

[Bokeloh et al. 2008] Bokeloh, M., Berner, A., Wand, M., Seidel, H.-P., Schilling, A.:
Slippage features
Technical Report, WSI-2008-03, University of Tübingen, 2008

[Bokeloh et al. 2009] Bokeloh, M., Berner, A., Wand, M., Seidel, H.-P., Schilling, A.:
Symmetry detection using line features
Computer Graphics Forum (Proceedings of Eurographics), 2009

[Bokeloh et al. 2010] Bokeloh, M., Wand, M., Seidel, H.-P.:
A Connection between Partial Symmetry and Inverse Procedural Modeling
In: ACM Transactions on Graphics (to appear), 2010

[Fischler et al. 1981] Fischler, M. A., Bolles, R. C.:
Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography
In: Communications of the ACM 24, 381-395, 1981

[Gelfand et al. 2004] Gelfand, N., Guibas, L. J.:
Shape Segmentation Using Local Slippage Analysis
Symposium on Geometry Processing, 2004

[Gumhold et al. 2001] Gumhold, S., Wang, X., McLeod, R.:
Feature Extraction from Point Clouds
In: Proceedings of 10th International Meshing Roundtable, 293-305, 2001

[Gumhold et al. 2001] Gumhold, S., Wang, X., McLeod, R.:
Feature Extraction from Point Clouds
In: Proceedings of 10th International Meshing Roundtable, 293-305, 2001

[Kazhdan et al. 2006] Kazhdan, M., Bolitho, M., Hoppe, H.:
Poisson Surface Reconstruction
In: ACM International Conference Proceedings, Vol. 256, 61-70, 2006

[Mitra et al. 2006] Mitra, N. J., Guibas, L. J., Pauly, M.:
Partial and Approximate Symmetry Detection for 3D Geometry
In: ACM Transactions on Graphics, 25(3), 560-568, 2006

[Pauly et al. 2008] Pauly, M., Mitra, N. J., Wallner, J., Pottmann, H., Guibas, L. J.:
Discovering Structural Regularity in 3D Geometry
In: ACM Transactions on Graphics, 27(3), 2008

[Yilmaz et al. 2009] Yilmaz, M. B., Erdogan, H., Unel, M.:
Probabilistic Facial Feature Extraction Using Joint Distribution of Location and Texture Information
In: Lecture Notes in Computer Science, Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II, 1171-1180, 2009