FACULTY
OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Symmetry Detection in Geometric Models

The State of the Art and Concept of Ph.D. Thesis

Lukáš Hruda

Technical Report No. DCSE/TR-2021-01
March 2021
Distribution: public

Technical Report No. DCSE/TR-2021-01
March 2021

Symmetry Detection in Geometric Models

The State of the Art and Concept of Ph.D. Thesis

Lukáš Hruda

Abstract

Symmetry occurs very commonly in real world objects as well as in artificially created geometric models. The knowledge about symmetry of a given object can be very useful in many applications in computer graphics and geometry processing, such as compression, object alignment, symmetric editing or completion of partial objects. In order to use the symmetry of any object in any given application, it first needs to be found. In this work, we provide some background about symmetry in general and about different types of symmetry, mainly in 3D objects. Then we focus on the task of automatic symmetry detection in 3D objects and we also describe the link between symmetry detection and the problem of registration. Most importantly, we present our own contribution in these fields. First, we show a new method of evaluating consensus in RANSAC surface registration together with a thorough analysis of various distance metrics for rigid transformations that can be used in this new approach. Afterwards, we provide an analysis of different representations of the space of planes in context of symmetry plane detection. At last, we propose a new, robust, fast and flexible method for symmetry plane detection based on a novel differentiable symmetry measure.

This work was supported by Ministry of Education, Youth and Sports of the Czech Republic, project PUNTIS (LO1506) under the program NPU I and University specific research project SGS-2019-016 Synthesis and Analysis of Geometric and Computing Models.
Copies of this report are available on
`http://www.kiv.zcu.cz/en/research/publications/`
or by surface mail on request sent to the following address:

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Acknowledgements

I would like to thank Prof. Dr. Ing. Ivana Kolingerová for patiently supervising my research and providing valuable council, and without whom this work would have never been possible. I further thank all my coworkers who cooperated with me on my research and also my family and my dear girlfriend for supporting me throughout my studies.

Contents

Chapter 1

Introduction

Symmetry is a potentially very useful feature which many real world objects exhibit. An object has symmetry if there is an operation or transformation (rotation, reflection, etc.) that maps the object onto itself, i.e. the object is invariant under the given transformation. The information about symmetry is instrumental in a variety of applications. It can be used for object alignment [77] where an object is properly aligned with the x, y, z axes using the symmetry information, or in compression where some parts of the object can be left out to lower the data size and then filled in during decompression using their symmetric counterparts [88]. Another application is symmetrical editing [60] which allows editing a single part of the object while the other symmetric parts are being edited in the same way automatically. Symmetry can also be used to reconstruct incomplete objects [97, 91, 61, 85, 89, 90] by filling the missing parts using the information from the symmetric counterparts. However, for any given geometric object, the symmetry information is not known in advance and needs to be first extracted which means that the symmetry first needs to be detected, ideally automatically by a computer. This is the task on which we focus in this work - automatic symmetry detection, i.e. creating algorithms that can be used to automatically find symmetry in geometric objects or models.

Symmetry detection in general is not an easy task. There is never perfect symmetry in real world objects. They only exhibit approximate symmetry and there is no strict way for a computer to decide whether some transformation captures an approximate symmetry or not. When detecting approximate symmetries the goal is mostly to find symmetries that appear natural to a human observer. See, for example, the two human faces in Figure 1.1. The one on the right was created by simply mirroring the one on the left and we can see that these two objects are quite similar so there is obviously some form of symmetry w.r.t. the mirroring operation. But we can also see that the symmetry is certainly not perfect, as can be expected since it is a 3D scan of a real human face and human faces never have perfect symmetry.

Although there are quite many methods for symmetry detection in geo-

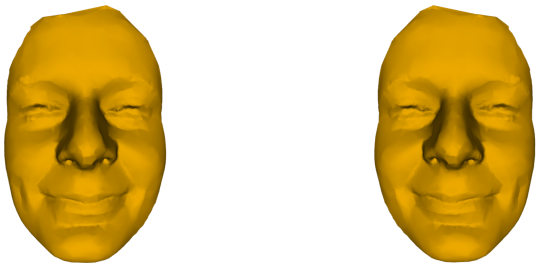


Figure 1.1: A 3D scanned human face [15] and its mirrored version.

metric data, there are still some challenges and a lot of room for improvement in the field of symmetry detection. Several different types of symmetry exist and geometric objects can have varying shapes and other properties. They can also be damaged in different ways, e.g. by noise or by having some parts missing. No method exists that could reliably and efficiently find symmetries of all the different types in any object. Even the methods that specialize in detecting symmetries of certain types, or a single type, can fail on certain types of objects or objects damaged in certain ways. Some methods only detect strong or perfect symmetries, some only work with specific object representations and some are simply not very fast. The main goal of this work is to enrich the field of symmetry detection by designing new more reliable and efficient methods or improving the existing ones to tackle the challenges of detecting weak, partial or more general symmetries efficiently and reliably and to make symmetry detection an easier task to solve in general. In some parts of this text we will also talk about the problem of registration which is strongly related to symmetry detection as it can be simply described as the problem of finding symmetry between two objects. Therefore, the registration problem also fits quite well into the topic of symmetry detection.

The rest of this text is organized as follows. Chapter 2 provides necessary background including definition of symmetry in geometric data and description of different symmetry types. Chapter 3 describes previous work in the field of symmetry detection and briefly in registration. In Chapters 4, 5 and 6 we present our own contribution to the fields of surface registration and relfectional symmetry (symmetry plane) detection in 3D objects including a whole new symmetry plane detection method. Finally, Chapter 7 provides conclusion and describes our intended future work in symmetry detection.

Chapter 2

Background

In this chapter we provide some background by defining symmetry in Euclidean data and describing different symmetry types in the 3D space. Then we also describe the task of registration which is strongly related to symmetry detection and we mention different representations of 3D objects.

2.1 Object and Symmetry Definition

We define a general object in an arbitrary Euclidean space E^d, where d is a positive whole number, in the following way.

Definition 1 *A set of points X is an object in E^d if $X \subset E^d$ and $X \neq \varnothing$.*

This means that we consider any non-empty set of points in E^d an object in E^d, regardless of whether the set is discrete or continuous. For example, an object in E^2 can be a 2D line or curve or any discrete set of 2D points. An object in E^3 can be a 3D line, 3D curve, a plane, a surface or a discrete set of 3D points.

It seems unclear whether a unique mathematical definition of symmetry is possible and whether all types of symmetry can be covered by a single definition [75]. However, symmetry of a geometric entity can be described as an invariance under a given geometric transformation. Therefore, for symmetry in d-dimensional Euclidean space we state a definition based on the one proposed in [75]:

Definition 2 *Having a function $F(\mathbf{x}), \mathbf{x} \in E^d$ and a transformation $T(\mathbf{x}) = \mathbf{y},\ \mathbf{x}, \mathbf{y} \in E^d$ then F has symmetry w.r.t. the transformation T if $F(T(\mathbf{x})) = F(\mathbf{x})$ for all $\mathbf{x} \in E^d$.*

To make this definition applicable to objects, considering an object $X \subset E^d$, we can simply set F as

$$F(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in X \\ 0 & otherwise \end{cases} . \tag{2.1}$$

Then we can alternatively define symmetry of an object in an arbitrary Euclidean space as follows.

Definition 3 *Given an object* $X \subset E^d$, $X \neq \varnothing$ *and a transformation* $T(\mathbf{x}) = \mathbf{y},\ \mathbf{x}, \mathbf{y} \in E^d$ *the object is symmetrical w.r.t. the transformation* T *if* $T(\mathbf{x}) \in X$ *for any* $\mathbf{x} \in X$.

In other words, T represents symmetry of X if, after applying T on any point of X, the point ends up in any other point of X or in itself, meaning the transformation maps the object onto itself. When F is set as in Equation (2.1) this definition is equivalent to the first and more general one.

This work elaborates on the task of automatic computer symmetry detection. Although the definition of symmetry in the previous section applies for objects in a Euclidean space of arbitrary dimension, we mostly focus on detecting symmetries in 3D objects. Therefore, unless stated otherwise, in the rest of this work we only consider symmetries and objects in E^3.

2.2 Symmetry Types

Symmetries in 3D objects can be classified based on different criteria some of which are described in this section.

2.2.1 Perfect and Approximate Symmetry

The symmetry definition in Section 2.1 is a definition of perfect symmetry where each point of the object is mapped exactly onto another point. However, perfect symmetry never occurs in real world objects and is quite rare even in artificially created digital models. In practice, objects only exhibit approximate symmetry where no transformation exists such that it maps the object onto itself perfectly but there might be transformations that map some portion of the object points "considerably close" to other points of the object. Therefore, when talking about symmetry detection in 3D objects, we usually mean detection of approximate symmetries, not perfect ones. Obviously, for approximate symmetries the conditions in the symmetry definition need to be relaxed in some way but no strict definition of approximate symmetry can be formed which in turn means that there is no exact way of deciding whether a transformation captures some approximate symmetry or not. This is what generally makes approximate symmetry detection quite a difficult task because we do not have an objective way for a computer to decide what is still perceived as approximate symmetry and what is not symmetry anymore. We will use the term *strong symmetry* for approximate symmetry that is close to perfect symmetry, and the term *weak symmetry* for symmetry farther from perfect.

2.2.2 Global and Local Symmetry

If an object has symmetry as a whole, it is called global symmetry. However, very often there is only symmetry in some parts of the object, i.e. a transformation exists that maps some part(s) of the object onto another part(s), which is called local symmetry. There is again no strict definition of local symmetry and both global and local symmetries are usually approximate in practice.

2.2.3 Symmetry Type According to Transformation Group

When detecting symmetries in 3D objects we are looking for transformations that represent these symmetries and these transformations always belong to a certain transformation group. Depending on the transformation group the symmetries in E^3 can be classified into several types.

Reflectional Symmetry

Reflectional symmetries (sometimes called mirror symmetries or bilateral symmetries) are represented by transformations from a group of reflection transformations, i.e. transformations that perform reflection over an arbitrary plane. A reflectional symmetry is usually and most easily described by a single plane of reflection. A plane that captures reflectional symmetry is called a symmetry plane. This type of symmetry is probably the most often occurring symmetry type in both real world and artificial objects which makes reliable methods for detecting symmetry planes in 3D objects quite desirable. For examples of reflectional symmetries see Figure 2.1. The objects are all rotated so that the symmetry plane, which is marked by the red line, is perpendicular to the image plane. Reflecting these objects over their symmetry planes would approximately map them onto themselves, therefore, these planes represent their approximate reflectional symmetries. The objects in Figure 1.1 are also examples of reflectional symmetry.

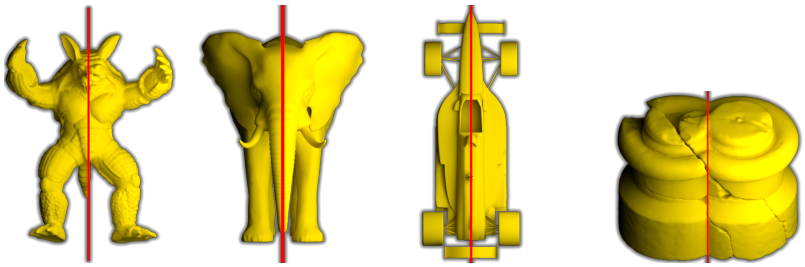


Figure 2.1: Examples of reflectional symmetry (figures taken from [42]).

Rotational Symmetry

Rotational symmetry is represented by a transformation that performs rotation by a given angle around a given axis. Such symmetry can be described by a rotation axis and a scalar value which defines the rotation angle. A special type of rotational symmetry is circular symmetry which can be represented only by the rotation axis where the symmetry holds for any rotation angle. A single circular symmetry basically represents infinitely many rotational symmetries. Some examples of approximate rotational and circular symmetries are in Figure 2.2. The rotation axis is marked by the blue line for each object.

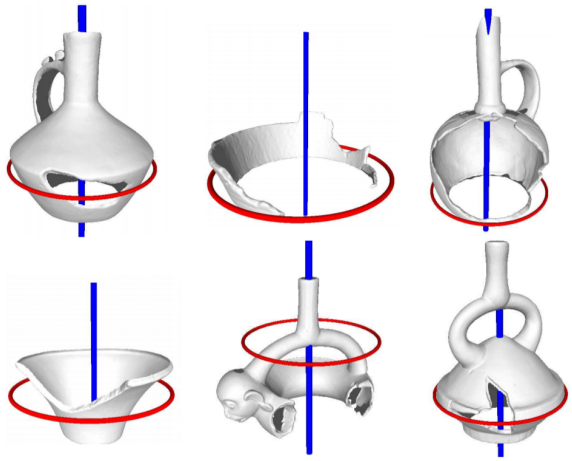


Figure 2.2: Examples of approximate rotational symmetry (figure taken from [89]).

Rigid Symmetry

Rigid symmetries are one of the most general symmetries that commonly occur in geometric models and they are represented by rigid transformations, i.e combinations of rotations and translations. Rigid symmetry can be described by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$. Often symmetry is considered rigid even if the transformation contains reflection and in such a case $\mathbf{R}$ can be either a rotation matrix or a product of a rotation matrix and a reflection matrix. When talking about rigid symmetries in 3D objects we often, but not always, consider local symmetries. Example of an object with local rigid symmetries can be seen in Figure 2.3, the symmetric parts are in the frames of the same color.

Other Symmetry Types

There are also other symmetry types such as point symmetry, which is represented by a reflection over a single point, or translational symmetry repre-

Figure 2.3: Example of an object with local approximate rigid symmetries (figure taken from [54]).

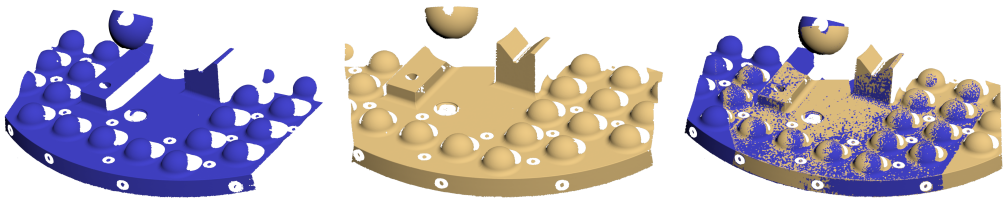
sented only by a translation. We can also consider more general symmetries represented by rigid transformations with additional uniform scaling or by general affine transformations. We could even have symmetries described by elastic transformations that can bend and stretch the object differently in different places or by transformations that perform reflection over an arbitrary surface or curve. If we do not put any constraints on the transformation type then we can basically for any object construct a transformation that captures its symmetry. These very general transformations are, however, very difficult to represent and have many degrees of freedom. Therefore, detecting such general symmetries is very hard.

Symmetries can be further classified as either extrinsic or intrinsic. Extrinsic symmetries are represented by geometric transformations in the Euclidean space. Until now we were only talking about extrinsic symmetries and all the symmetry types we have mentioned are types of extrinsic symmetry. Intrinsic symmetry is rather specific and does not really satisfy our definition of symmetry, but, just for completeness, we mention it anyway. Intrinsic symmetries are specific to surfaces and are described by intrinsic transformations on the surface rather than extrinsic transformations in the Euclidean space in which the surface is embedded. Usually intrinsic symmetries are defined as transformations that maintain geodetic distances between all points on the surface. Such symmetries are invariant under deformation of the surface, such as bending or folding, as long as the deformation does not change the geodetic distances of any points. There are several methods that are somewhat capable of detecting intrinsic symmetries, e.g. [99, 69, 100], however, this area is distinct from the focus of our research and by finding a good intrinsic symmetry, we do not necessarily get a meaningful extrinsic symmetry of the desired type. Also, intrinsic symmetries are represented by point-to-point or part-to-part correspondences which are rather impractical representations for extrinsic symmetries, which can usually be represented by simple geometric transformations. Therefore, in the rest of this work we will not be considering intrinsic symmetries in any way.

2.3 Registration

Registration is a field strongly related to symmetry detection. Having two input objects, the task of registration is to find a transformation that maps one of the two objects onto the other one. So, only instead of mapping an object onto itself we are trying to map an object onto another object, therefore, registration can basically be understood as detecting symmetry between two objects. Another way of seeing this is that symmetry detection is a special case of registration where we are registering an object onto itself (the two input objects are the same), i.e. that registration is a more general extension of the symmetry detection task. Either way, symmetry detection and registration have very much in common and many principles and approaches used in registration methods can be also used in symmetry detection and vice versa.

Therefore, given the relation between these two areas, we are also going to consider registration in some parts of this work. Specifically, we will talk about rigid surface registration which is a task of finding a rigid transformation that maps a 3D surface onto a different 3D surface in such a way that some portion of the two surfaces overlaps. This is usually used for completing objects from partial 3D scans. An example of a correct registration can be seen in Figure 2.4.



(a) First surface (b) Second surface (c) Correct registration

Figure 2.4: Example of a correct registration - (c) of the two surfaces - (a), (b) (figure taken from [37]).

2.4 Object And Other Data Representations

When trying to detect symmetries in 3D objects, we first need to define what representation of an object will be on the input. Throughout this text, we will work with two different representations of general 3D objects - a discrete point set (point cloud) and a triangle mesh. Here we describe these representations and also briefly mention other representations of objects and of some non-object geometric data.

2.4.1 Discrete Point Set or Point Cloud

A discrete point set, often called a point cloud, is very commonly used in geometry processing to represent a sampled 3D surface but it can also represent a sampled volume. Using the discrete point set representation, any 3D object X can be simply defined as

$$X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}, \mathbf{x}_i \in E^3, i = 1, ..., n$$

where n is the number of the points in the point set. Discrete point sets are usually outputted e.g. by 3D scanners.

2.4.2 Triangle Mesh

The triangle mesh representation is very commonly used in computer graphics to describe a surface. A triangle mesh is often understood as an approximation of a continuous surface by a piece-wise planar surface where the planar pieces are triangles. Having an object X, to use the triangle mesh for its representation we need a set of vertices $X_v = \{\mathbf{x}_i \in E^3\}, i = 1..\nu_X$ and a set of triangles $X_t = \{t_i^X\}, i = 1..\tau_X$. For each triangle $t^X \in X_t$ three indices $k1$, $k2$, $k3$ are stored and the triangle is defined by the three vertices $\mathbf{x}_{k1}$, $\mathbf{x}_{k2}$, $\mathbf{x}_{k3} \in X_v$. The object is then defined using the triangle mesh representation as

$$X = \bigcup_{i=1}^{\tau_X} t_i^X$$

i.e. it is a union of all the triangles. We could, of course, only have the triangles without the vertices while describing the same exact object. This, however, would not be a triangle mesh but only a triangle set. Unlike the triangle set, the triangle mesh allows easily extracting adjacency information which is required for many operations, such as traversal or estimating differential quantities (e.g. curvature), which would be much more difficult to perform without it.

The triangle mesh is a more general representation of an object than the discrete point set because it contains more information. A discrete point set can always be easily derived from a triangle mesh simply by taking its vertices or using some surface sampling technique. But transforming a discrete set of points into a triangle mesh is not so straightforward and is considerably more difficult. This also means that methods that are capable of working with the discrete point set representation are generally more widely applicable than those that require a triangle mesh on the input.

2.4.3 Plane Representation

In some parts of this text we will also work with planes in E^3, therefore here we describe how to represent them. Unless stated otherwise, we define an

arbitrary plane P by its implicit equation $P : ax + by + cz + d = 0$ where $\mathbf{x} = [x, y, z]^T \in E^3$ is a general 3D point, a, b, c, d are the coefficients that define the plane and the points $\mathbf{x} \in E^3$ that satisfy this equation are also the points of the plane P. The vector $[a, b, c]^T$ describes the normal of the plane and the d coefficient encodes the distance of the plane from the origin of the coordinate system. This representation is probably the simplest one but it is also ambiguous since for any real number $k \neq 0$, the coefficients a, b, c, d define the same plane as coefficients ka, kb, kc, kd.

2.4.4 Others

General objects in E^3 can also be represented in other ways, e.g. as parametric or implicit surfaces, parametric curves or binary functions which can represent volumes by having a value of 0 for points outside the object and 1 for points inside the object. Objects can also have values associated with each of their points. These values can be scalars or vectors and they can represent various quantities, such as colors or results of some physical measurements (temperature, pressure, etc.). When considering symmetries of such objects, sometimes, but not always, it is desired that apart from the positions of the points these associated values are also taken into account. In such a case Definition 3 of (perfect) symmetry of objects is no longer applicable and the more general Definition 2 needs to be used with the function F defined accordingly.

There are also other types of geometric data that do not satisfy our definition of an object but appear commonly in certain areas of computer science. We can have a general voulmetric function which associates a value with each point in E^3 or in some subspace of E^3. Or we can have a discrete volumetric grid where each cell has a value assigned to it - this is mainly used to represent medical images outputted by CT or MRI scanners. For such data we also need to use Definition 2 when talking about symmetries.

However, the representations mentioned in the previous two paragraphs will be used in this work only rarely.

Chapter 3

Related Work

In this chapter we provide information about various existing symmetry detection methods. We mention some only briefly while some, which seem more important or interesting, will be described in more detail. Since the methods are quite diverse and based on various approaches it is quite difficult to find some way to classify them in order to make the chapter easier to read. Therefore, we divide the methods only according to the type of symmetry they are designed to detect. At the end of the chapter we also briefly describe several existing methods for rigid surface registration. Some parts of this chapter were taken from the author's master thesis [35].

3.1 Reflectional Symmetry

The most common type of symmetry being solved for 3D objects is reflectional symmetry. Such symmetry is captured by a single plane of reflection and there are many strategies of computing such a plane.

Some methods detect only planes that pass through some reference point such as the origin, centroid of the input object or its center of mass. Such methods are generally not very good at detecting weaker symmetries because weakly symmetrical objects can have the symmetry plane in an arbitrary position, not passing through the expected reference point.

Sun and Sherrah [94] proposed symmetry plane detection using orientation histogram and their method detects only planes that pass through the origin. The method uses the discrete version of the Extended Gaussian Image [34] called orientation histogram. It is obtained by dividing a unit sphere into hexagonal bins with values assigned according to the number of normal vectors of the input object facing in the given bin's direction. Several candidate planes are selected such that they pass through the origin and their normal vectors are facing approximately in the directions of the principal axes of the object. For each candidate the histogram is reflected over the given plane and a correlation of the reflected histogram with the original

one is computed. The plane with the highest correlation is then selected as the strongest symmetry plane. The method can also detect rotational symmetries with rotation axes passing through the origin but the algorithm is slightly different.

Kakarala et al. [45] proposed a method that uses approximation of the input object with spherical harmonics, creating a star-shaped surface, and the symmetry plane detection is then performed on this new shape. The symmetry detection is based on the observation that if a real-valued function has symmetry across the origin then its Fourier transform is real-valued. The authors apply this observation on the spherical harmonics to derive an error function whose optimization leads to the symmetry plane. This method only detects planes that pass through the origin.

Korman et al. [46] designed a method that uses a distortion measure representing the amount of mismatched volume between the original shape and the transformed shape. The method detects only planes that pass through the origin but it can also detect rotational symmetries (with axes also passing through the origin). Since the evaluation of the distortion measure is too complex, the authors propose a randomized sampling procedure that gives approximately the same results with overwhelmingly high probability. The symmetry of an input object is than found by clever sampling of the transformation group based on the observation that the upper bound on the sampling density can be controlled by the maximum allowed distortion of the symmetry. The distortion is evaluated in each sample and the one with the lowest distortion can be selected as the strongest symmetry.

Li et al. [53] proposed a view-based method for symmetry plane detection on triangle meshes. A set of viewpoints is sampled on a sphere around the input model and in each viewpoint a camera is set to look at its center. Using these cameras the model is rendered from each viewpoint using orthogonal projection. The authors employ a viewpoint entropy which can be computed for each rendered view and depends on the areas of the faces that were rendered in the particular view. This entropy is computed for each viewpoint which creates a viewpoint entropy distribution sphere. The symmetry plane detection itself is then based on the observation that the symmetry planes of the model and of the viewpoint entropy distribution sphere are the same. Therefore, the symmetry plane detection is performed on the viewpoint entropy distribution sphere which has the advantage of being independent on the vertex count of the input mesh. Candidate planes are created using pairs of viewpoints with matching values of the viewpoint entropy. For each candidate plane the rest of all pairs of viewpoints with matching entropy are verified to see whether they are symmetric with respect to the given plane. If the number of symmetric pairs is great enough the given candidate plane is declared a symmetry plane. This method only

detects planes passing through or very near the center of the input object's bounding sphere. However, it seems to outperform previously designed methods in terms of accuracy, speed and robustness to noise.

The following methods do not put any constraints on the detected planes and most of them can be used for detecting weaker symmetries.

Schiebener et al. [85] designed a method which detects the plane of symmetry of a point cloud and they use it for completion of partial 3D scans. Apart from the point cloud representing the input object, the method also needs a point cloud representing the object's surrounding and the position from where the object was scanned. It relies on the fact that a 3D object usually stands on some kind of supporting structure, such as the ground, and the first step is detecting the supporting plane which represents such a structure. Candidates for these supporting planes are created using a RANSAC-based plane fitting algorithm and for each of these candidates, several symmetry plane candidates are created by sampling the space of planes orthogonal to the given supporting plane. The authors define rating of the candidate symmetry planes which uses the supporting plane and the scanner position to evaluate whether points of the input object, when reflected over a given candidate symmetry plane, end up in plausible locations. For example, the reflected points should appear above the supporting plane and mainly in occluded areas. The plane with the highest rating is declared the object's plane of symmetry. This method only detects symmetry planes which are orthogonal to the supporting surface and, because it requires the surrounding and scanner position, it is only usable in specific applications.

Simari et al. [88] presented an algorithm for detecting local reflectional symmetries of 3D triangle meshes. First, a weighted covariance matrix and a weighted centroid are computed, with the weights set for each vertex according to the areas of its adjacent triangles. Eigenvectors of the matrix are used to define three orthogonal planes that pass through the centroid. For each of the planes all vertices are reflected over it and its cost is computed based on the distances of the reflected vertices from the original mesh (using the minimum point-to-triangle distance). Only the plane with the lowest cost is kept and a support region is defined as the region of the mesh which has a strong enough symmetry w.r.t. to this plane. Next, the weights of all vertices are updated to contain information of how far from the mesh the given vertex is when reflected over the plane - the farther, the smaller its weight. Also, weights of all vertices outside of the support region are set to 0. A new covariance matrix and a new centroid are computed using the new weights, a new plane is estimated and a new support region is found. These plane estimation and region finding steps are iterated until convergence is achieved. The vertices in the support region of the final plane are removed and the whole process is repeated for the remaining components of the mesh. This

way a tree structure can be created called a folding tree which the authors propose to use for mesh compression.

Podolak et al. [77] proposed a Planar Reflective Symmetry Transform (PRST) for volumetric functions which associates each plane in the space of planes with a value determining how symmetrical the function is w.r.t. that plane. This value is defined as the distance of the function to the closest function which is perfectly symmetrical w.r.t. to the plane. The transform can be used to find the plane of symmetry of a discrete volumetric function represented by a uniform 3D grid. The brute-force computation of the PRST has time complexity $\mathcal{O}(n^6)$ for a grid of $n \times n \times n$ voxels but the authors propose a way of finding the plane with the highest PRST in $\mathcal{O}(n^4 log(n))$. To use this approach for symmetry detection on a sampled surface (triangle mesh, point cloud) the authors use the Gaussian Euclidean Distance Transform to convert the surface to a volumetric grid. To lower the computation cost for the surface data, the authors propose using a randomized Monte-Carlo algorithm to approximate the PRST. The space of planes is divided into discrete bins based on their normal orientation and distance from the origin. Pairs of points are randomly being selected, for each pair the symmetry plane of the two points is constructed and it votes for the plane represented by the corresponding bin. The planes with the largest vote count represent the most significant symmetry planes of the input object. Since these results are not necessarily very precise, the authors propose a final iterative refinement. Although this approach works with sampled surface data, it still requires rasterizing the input object, which can be computationally expensive and seems unnecessary.

Cailliere et al. [13] used Hough transform to detect symmetry planes of triangle meshes. This method is actually a modification of the more general clustering approach [67] that will be described in Section 3.3. It is overall also quite similar to the Monte-Carlo approach used in the previous method by Podolak et al. [77]. Points are paired based on similar curvature values and for each pair its symmetry plane is constructed. These planes vote for the planes represented by the corresponding bins in the space of planes and, in the end, the plane with the largest point count is selected as the symmetry plane of the input mesh.

Speciale et al. [93] employ two symmetry plane detection methods for 3D shape reconstruction. Both the methods take depth information in voxel grid as input and one of them is essentially the implementation of the above described method of Podolak et al. [77]. The second method is a little different. High-gradient and high-curvature voxels are extracted and referred to as surface voxels. Pairs of surface voxels are randomly sampled and for each pair a symmetry plane candidate is constructed. Each candidate is evaluated according to the number of inliers. A surface voxel is considered

an inlier if, when reflected over the given candidate plane, it ends up in another surface voxel or in space with unknown voxel information where a potential occluded part of the symmetric object could be. Planes with the largest inlier count then represent the strongest symmetries. The authors claim that this approach, although similar to [77], can potentially be faster and has lower memory requirements than [77].

The methods of **Combès et al.** [23] and **Ecins et al.** [25] both use iterative ICP-like (see Section 3.4 for ICP) approaches for symmetry plane detection in point clouds. Such approaches can be roughly summarized as follows. Initial plane or planes are estimated in some way and for a given initial plane and iterative process is executed. In each iteration points of the object are reflected over the plane and those that reflect too far from other points of the object are usually considered outliers and ignored in that iteration. For each of the remaining reflected points, the closest point of the object is found and a correspondence between the two points is created. Next, the plane is adjusted such that the corresponding pairs of points have the best possible reflectional symmetry w.r.t. the new plane - least square error is usually used for this purpose. These steps are iterated until convergence and the final plane usually represents some significant symmetry of the input object.

Sipiran et al. [91] proposed a method specifically designed for global symmetry plane detection on incomplete objects represented by triangle meshes. First step is to detect local features of the given 3D shape which is done using the theory of heat diffusion on manifolds. A function is defined which associates the accumulation of heat up to a given time to each point on the surface. This function is computed using the eigenvalues and eigenvectors of the Laplace-Beltrami operator and is called a Heat Kernel. As the feature points, local maxima of the heat accumulation function are taken. Pairs of the feature points then generate the candidate symmetry planes but only pairs of points which have a similar value of the heat accumulation function are considered. The last stage of this algorithm is a voting process where other pairs of points are tested against the candidate planes. The more pairs of points are considered to be symmetrical with respect to a given plane the more votes the plane gets. There are several criteria designed by the authors, which are used to decide whether or not a plane is considered a plane of symmetry of a given pair of points. Also only points in which the mean curvature is larger than some threshold are used in the voting process. In the end the plane with the highest vote count can be declared the resulting plane of symmetry. This method seems to provide very good results even when used on objects with very high level of missing parts. However, it only works on manifold triangle meshes and it does not work on featureless objects, which is quite constraining.

Cicconet et al. [19] solve the symmetry plane detection problem as a registration problem. They propose reflecting the input object over a fixed plane and then using some rigid registration algorithm to map the reflected object onto the original one. This approach results in a transformation that represents a reflection over some plane followed by some rigid transformation. Such a transformation does not generally represent reflection over a plane, so it needs to be extracted approximately from it, which seems unnecessarily more complex than finding the reflection plane directly. Furthermore, as elaborated in [80], the registration algorithms sometimes do not work well on symmetrical objects.

Most recently, **Nagar and Raman** [70] proposed using linear pairing and optimization on manifold to find the symmetry plane of a point cloud. This method works in a space of arbitrary number of dimensions, but due to time complexity $\mathcal{O}(n^{3.5})$ for n points, it is considerably slow. The same authors then proposed a closed form solution to the symmetry plane detection problem with randomized initialization [71]. However, at some stages this method expects the symmetry plane to pass through the center of mass which, as mentioned above, could possibly be problematic for some weakly symmetrical objects.

There are also methods that approach symmetry plane detection by employing machine learning techniques, usually using neural networks. One of the newer representatives of this approach is e.g. the method of **Ji and Liu** [44]. Such methods naturally require a set of training data and a training process to be executed before any symmetry detection can be performed which considerably limits their applicability.

3.2 Rotational Symmetry

Some of the methods mentioned above can also be used to detect the axes of rotational symmetries, e.g. [94, 46], but they exhibit the same problems and limitations for this purpose as for the symmetry plane detection.

Martinet et al. [60] designed a method for detection of rotational symmetries with possible reflections in triangle meshes using generalized moment functions. It is based on the observation that the generalized moments of a shape have at least the same symmetries as the shape itself and the symmetries of the moments can be computed efficiently. This method only detects symmetries with rotation axes passing through the input object's center of mass but it can also find pure reflectional symmetries (symmetry planes).

Sipiran recently proposed two methods [89, 90] for rotational symmetry detection based on finding circular structures in the data. However, they only

seem to detect symmetries of rotational shapes and do not detect discrete symmetries where the symmetry only holds for certain angles of rotation, i.e. they can only detect circular symmetries.

3.3 More General Symmetry

There are also methods for detecting more general symmetries.

Thrun and Wegbreit [97] designed a method for detecting reflectional symmetries, point symmetries, some types of rotational symmetry and several composites of these symmetry types. The method is designed specifically for the purpose of object reconstruction in computer vision and requires the knowledge of the position from which the input object was scanned because it needs to know which areas of the space are occluded.

Bokeloh et al. [9] proposed a method for finding rigid symmetries in point sets with normal vectors available for each point. First, the method finds feature line segments in the input object using so called slippage analysis [29] together with the normal vectors and mean curvature estimates. Line segments that are spatially close are then connected to build a feature graph. Lines that lie on the same circular arc are grouped together and form a line cluster - a longer piece of a line feature with constant curvature. Pairs of line clusters that are connected by an edge in the feature graph are called bases, each basis defines a local coordinate system. Candidate symmetries are constructed by matching bases using curvature information of the line clusters and the underlying geometry, the symmetries are scored and only the best ones are kept. A geometric validation is performed in the end to verify the symmetries and establish correspondences of the symmetric parts. This method naturally requires significant features to be present in the input data in order the find any symmetries.

Lipman et al. [57] detect rigid symmetries in a point set using a symmetry correspondence matrix. The value at the position i, j in the correspondence matrix quantifies how much the i-th and j-th points of the input point set belong to the same orbit. Two points are in the same orbit if a symmetry transformation exits that takes one of the two points to the other one. The symmetry correspondence matrix is approximated by random sampling where each sample consists of two pairs of points that satisfy certain criteria (e.g. that the two points have similar distance in both pairs or that the pairs are not too close to each other). A local coordinate frame is constructed for each of the two pairs using both points in the pair together with an approximated normal in the first point and a transformation is constructed that aligns the two coordinate frames. This transformation is then applied on the entire point set and the deviation from the original point

set is measured. The deviation is then cast into the values in the matrix that correspond to pairs of points which the transformation brings close to each other. The authors state several observations about the eigenvalues and eigenvectors of the symmetry correspondence matrix (e.g. that the number of non-zero eigenvalues i equal to the number of orbits) which are used to extract the symmetry information about the input point set. This method is overall quite generic but it can run minutes on objects with only hundreds of points and its parameters need to be set manually.

Mavridis et al. [61] proposed a method for detecting rigid symmetries on objects with missing parts and they use the symmetries for object reconstruction. They solve the symmetry detection problem by maximizing the overlap of the object and its transformed version with an additional constraint that some portion of the two objects does not overlap. This constraint is added because this method explicitly expects an object with some missing parts on the input. To achieve this goal, the authors use the Super4PCS [63] rigid registration method (see also Section 3.4) with the additional modification that at least k percent of the points of the two objects should not be overlapping. With $k = 0$ the task becomes an ordinary registration of the object onto itself which would probably always output the identity as the resulting symmetry. This method obviously stands on the quality and properties of the selected registration method.

Li et al. [54] use co-occurrence analysis to find rigid symmetries. They use a curvature-based descriptor to compute features for points of the input object but they state that the overall method is independent of the specific descriptor design. Next, clustering in the feature space is used to find points with similar features, creating clusters of feature values. For each pair of these feature clusters a transformation is estimated that best matches the set of points that correspond to the first feature with the points that correspond to the second feature. A probability that the two features co-occur is computed based on the number of points that are transformed close to points from the other set. The co-occurrence information is embedded into a low-dimensional space where co-occurring features are close to each other and clustering is used to find modes in this space. These modes then represent building blocks of the input object - repetitive patterns of which the input object consists. Instances of these blocks are naturally linked by symmetries. As the authors state, this method does not work well for objects with noise, irregular sampling or objects without significant features where the descriptor fails.

Probably the best known and most popular method for symmetry detection was designed by **Mitra et al.** [67] who used clustering in the transformation space for detecting very general symmetries. The symmetry transformations can contain rotation, with or without reflection, translation and

even uniform scaling. The method can also be restricted to only detect subgroups of these general symmetries. The input object is sampled and in each sampled point, the principal curvatures and principle directions are computed creating a local coordinate frame together with the normal vector. Candidate transformations are created by pairing the sampled points and aligning their local frames, but, since points where both principal curvatures are the same do not define unique transformations, such points are excluded beforehand. The scale component of each transformation is estimated from the ratio of the principle curvatures in the two points. If the two generating points of the candidate transformation have too distinct signatures (represented by the curvature values), the transformation is discarded to ensure that only candidates that approximately match the local patches of the two points are kept. Since the candidate transformations map local patches onto each other, each of the transformations provides evidence of symmetry. The core idea of the method is that now the transformations that represent significant symmetries will appear most frequently forming modes in the candidate set, so the symmetry detection problem is approached by mode-seeking in the transformation space. Of course, the transformations in the modes are generally not exactly the same, only similar, therefore, the mode-seeking is solved by clustering. Specifically the mean shift clustering algorithm [22]. The detected clusters then represent the symmetries of the input shape.

Much later, **Shi et al.** [86] improved this method by designing and employing a more appropriate metric in the transformation space during the clustering stage. They also use a slightly different criterion for deciding what candidate transformations to discard. Apart from this, the method [86] is almost identical to [67].

Although these clustering-based methods are very general, they are not very robust to noise and our experiments also suggest that they are rather sensitive to parameter setting.

There are a few more methods that we do not describe in detail but are worth mentioning. A method for solving a symmetry-related problem of finding regularities in 3D data was proposed by **Pauly et al.** [74]. Some of the steps in this method are similar to certain steps of the previously mentioned method by Mitra et al. [67]. **Tevs et al.** [96] used a method for detecting rigid symmetries that is inspired by Pauly et al. [74] and Bokeloh et al. [9]. **Xue et al.** [102] constructed a method for finding affine symmetries but it is designed particularly for architectural data which are very specific. **Alcázar et al.** [3] proposed a method for detecting quite general symmetries in rational space curves and **Hauer et al.** [31] find affine and projective symmetries but only for rational and polynomial surfaces and their method only seems to detect very strong or perfect symmetries.

3.4 Rigid Surface Registration

In this section we very briefly mention some methods that attempt to solve the problem of rigid surface registration where the goal is to map one surface onto another using a rigid transformation. The text in this section was mostly taken from our paper [37].

The registration process can be classified as either local registration or global registration, but in this case the terms local and global have a slightly different meaning than in context of symmetry. While a local registration only improves some initial alignment and makes it more precise, a true global registration finds an alignment without any assumptions on the initial position of the inputs. Typical registration pipeline consists of first using a global registration method to find a rough alignment and then employing some local registration method to improve its accuracy.

Perhaps the best known approach to local rigid surface registration is the Iterative Closest Point (ICP) algorithm [17, 8], and its derivatives [83, 78, 10] and generalizations [66]. The algorithm improves a given alignment by searching for point correspondences, usually based on local proximity, color similarity or other point properties. Based on the set of correspondences, an optimal rigid transformation is found that maps the points from the first object to their corresponding counterparts in the second object. These steps are iterated until convergence. Given a good initialization, the algorithm converges quite quickly, making it a good choice for local registration.

Algorithms for global registration are usually designed to be fully independent of the initial position of the inputs. Some algorithms decouple the rotation from translation, searching for the optimal rotation first. One possibility is using the Hough transform [5, 14]. Spherical maps of the input surfaces are created, where each point represents a direction, in which a certain property of the input data is computed. A rotation is then sought which maps the two spherical maps one onto another. Having the rotation, translation is found by projecting the rotated inputs onto coordinate axes and aligning the projections in each axis separately.

Another possibility of decoupling the rotation is to use the Extended Gaussian Image [34], approximated by a spherical histogram of normal orientations. Two such histograms can again be aligned first in the space of rotations [59], while the translation is found afterwards.

Another group of methods attempts to extend the notion of Phase Correlation [24], which has been successfully used for image alignment, to 3D point clouds. Usually, the input objects are resampled onto a regular grid, which is then transformed into frequency domain, where the aligning rotation can be found [12]. Unfortunately, the extension only works in a certain range of angles, and thus it cannot be applied for a general global registration.

Concepts of evolutionary algorithms were also applied to searching for the best aligning transformation [11, 18]. An initial population of possible alignments is created, and a complex evaluation function is used to determine the fitness of each phenotype. The most successful phenotypes are recombined and mutated to create the next generation of alignments. The method usually delivers a good match, however, its computational complexity is high.

Another global registration method has been proposed based on searching for pairs of congruent planar quadrilaterals in both input datasets [2], known as 4PCS (4-point congruent set). The method builds on a smart tree data structure which allows finding the congruent quadrilaterals based on their midpoint, called invariant, efficiently. An improved version of the method, the Super4PCS [63], eliminates some of the quadratic steps, such as finding all pairs of vertices at given distance, by using another spatial structure built in a preprocessing step.

The Fast Global Registration [106] stands on the border between local and global approaches, despite its name. It uses Fast Point Feature Histogram (FPFH, [84]) features that are matched and a subset of valid matching pairs is iteratively refined, while each iteration also improves the aligning transformation.

Naturally, some of the concepts and approaches used in the methods for symmetry detection that were described above could also be utilized in surface registration. On the other hand, some of the registration methods mentioned in this section could quite likely be modified to perform some form of symmetry detection as done e.g. in [61] or [19].

Chapter 4

Consensus Evaluation in RANSAC Surface Registration

In this chapter we describe our contribution to the field of rigid surface registration. The task of rigid surface registration is, having two surfaces on the input, find a rigid transformation (i.e. combination of rotation and translation) that best aligns the first surface with the second one. As already mentioned in Section 2.3, rigid surface registration is a field strongly related to symmetry detection since it can be understood as detecting rigid symmetry between two objects. The contribution and results presented in this chapter can therefore very easily find its application in the area of symmetry detection as well. The content of this chapter was previously published in [37].

Several registration algorithms mentioned in Section 3.4 (among others [59, 2, 14, 63]) can be interpreted as a particular implementation of the general RANSAC (Random Sample Consensus) approach, which is very commonly used in surface registration and other areas including symmetry detection. The overall scheme of a general RANSAC registration algorithm is following. By estimating the correspondences of points of the first surface with points of the second surface, a set of candidate transformations is constructed, each of which succeeds in aligning at least a part of the input surfaces. Then, the consensus of the candidate transformations is evaluated to pick the best one. Some algorithms, including the current state of the art Super4PCS [63], seek consensus of points of the two input surfaces w.r.t. the candidate transformations. The candidate transformations are being applied on the first input surface and the consensus is determined for each candidate as the portion of the transformed surface that overlaps with the second surface. Generally, the candidate with the largest overlap is selected as the best one. However, we argue that this might not be the best possible approach.

The key problem of finding the consensus this way stems from the difficulty of evaluating the result. Usually, the user is expected to define a certain distance δ, and the quality of the alignment is defined as the size of

the overlap, i.e. the portion of the input meshes that get mapped closer to each other than δ. Choosing a large δ leads to false matches, which align incorrect parts of the two objects. On the other hand, choosing a small δ leads to a narrow global optimum, which can be missed even with a rather dense sampling of the solution space, and a large number of local optima which complicate iterative refinement. For practical registration, δ cannot be determined universally, not even as a proportion of the input scale, as we demonstrate in our experiments. Moreover, since the solution involves an estimate of a rotation, it is possible that remote parts of the surface become severely misaligned, even when the rotation estimate is only off by a few degrees.

As a remedy to this problem, we propose *using information from all the candidates* to better analyze the space of rigid transformations. The key observation is that the optimal alignment *forms a density peak in the space of rigid transformations*. The problem can therefore be solved using some clustering algorithm (k-means, mean shift), interpreted as a facility location problem, or solved by efficiently estimating the *density* at each candidate location. In our experiments, the last choice is both the fastest and the most reliable. Similar idea was previously used by Mitra et al. for symmetry detection [67] and here we show how a related approach can be employed in surface registration.

In order to perform a clustering or a density estimation, a crucial ingredient is a *metric* that relates the elements of the given space, i.e. determines the amount of (dis)similarity of rigid transformations. This turns out to be a non-trivial problem, because the metric has to relate the different degrees of freedom of a rigid transformation (rotation, translation) to obtain a single distance value. This relation depends on the units used for translation and rotation, as well as on the scale of the object: for large objects, a unit rotation (for example one degree) has a bigger impact than a unit translation, and vice versa.

There are several well defined metrics for rotations [43], and the distance of two translations can be simply defined as the Euclidean distance of the two translation vectors. However, the problem of computing distances of combined transformations seems insufficiently addressed. In the related context of symmetry detection Mitra et al. [67] used the mean shift clustering algorithm with a metric which uses a weighting scheme that imperatively relates a rotation by π radians to a translation by one half of the body diagonal. In [92] the authors attempt to address the problem of measuring distances of 2D transformations by searching for geodesics on the manifold of rigid transformations and similar idea is used by [86] for transformations in 3D, however, the problem of proper scale is not addressed. A metric in $SE(3)$ is used by [6] to generate smooth transformations. The problem of relating rotations and translations is bypassed by inheriting a metric from a higher-dimensional manifold $GL(3)$.

A common approach to constructing a metric for rigid transformations is to compute it as a weighed sum of some rotation metric and the Euclidean translation metric [50]. In this chapter, we analyze this approach and show that, although it is well applicable, it needs to be used with caution, because of several problems inherent to such metrics that may not be obvious. We also show how these problems can be mitigated or even eliminated using a different, data-induced transformation metric which embraces a different interpretation of the problem, and we demonstrate that in order to properly relate the rotation and translation component, the difference between two rigid transformations *necessarily must depend on the input data, and should be evaluated as such*. A metric in $SE(3)$ that fulfills this criterion has been defined in [79] to evaluate registration results, however, it has not been used in the registration (nor symmetry detection) process itself.

As a model case, we study a generic global RANSAC-based rigid surface registration scheme. We compare several metrics in the context of this algorithm, as well as different consensus evaluation strategies, and we report the results on multiple non-trivial registration datasets.

Our contribution is threefold:

- we analyze a global registration algorithm that is based on creating a set of candidate alignments, followed by finding a density peak in the space of rigid transformations using an efficient data structure - the Vantage Point Tree,
- in the context of the algorithm, we test a variety of distance metrics, examining their properties theoretically and experimentally,
- we propose certain modifications to the compound metrics that improve their performance in the task at hand and potentially in other applications as well.

4.1 Model Registration Algorithm Description

As input, we have two triangle meshes P and Q with vertices $\mathbf{p}_i, i = 1..\nu_P$ and $\mathbf{q}_i, i = 1..\nu_Q$, and triangles $t_i^P, i = 1..\tau_P$ and $t_i^Q, i = 1..\tau_Q$. The result is a rigid transformation (an orientation-preserving isometry) that aligns Q to P. Since neither P nor Q represent the entire object, the overlap is only partial.

The model algorithm consists of the following steps:

1. A subset of the vertices of P is selected, so that their distribution is roughly uniform. For each sample, the principal curvatures κ_1, κ_2, as well as the first principal direction $\mathbf{e}_1$, are estimated and stored.

2. The mesh Q is also uniformly sampled, and for each sample, the best matching sample in P is identified based on the similarity of principal curvature estimations. A rigid transformation is constructed, which maps the corresponding points, and its quality is evaluated. High scoring transformations are stored for further processing.

3. The set of candidate transformations is interpreted as a point set in the SE(3) space, and a density peak that can be interpreted as a candidate consensus is sought.

In the following, we describe each step in more detail.

4.1.1 Curvature Sampling

First, we need a feature vector that describes the local shape of the objects. It is possible to use various descriptors proposed in the literature [33]. For the purposes of our model scenario, we need a descriptor that is fast to evaluate, yet considers larger local neighbourhood than just a 1-ring in order to be resilient to noise. We choose to use a very simple custom local feature vector, only consisting of the two principal curvatures estimated from points in a local neighborhood, which is found using a breadth-first search (BFS) of the mesh connectivity. We stop the BFS at a certain Euclidean distance from the sample point.

Having a set of N neighbors $\mathbf{v}_j, j = 1..N$ of a vertex $\mathbf{p}_i$, we estimate the shape operator in a way similar to [82]. In contrast to Rusinkiewicz's method, we do not attempt to obtain exact curvature, and thus we can omit some steps in order to simplify the computation. We parameterize the tangent plane, defined by the normal $\mathbf{n}_i$ of the vertex $\mathbf{p}_i$, with a local othonormal basis $(\mathbf{l}_x, \mathbf{l}_y)$ (see Fig. 4.1). We then use the property of the shape operator that for a tangential position difference vector it yields the corresponding change in normal, i.e. for each neighbor $\mathbf{v}_j$ the following relationship represents two linear equations:

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} (\mathbf{v}_j - \mathbf{p}_i) \cdot \mathbf{l}_x \\ (\mathbf{v}_j - \mathbf{p}_i) \cdot \mathbf{l}_y \end{bmatrix} = \begin{bmatrix} (\mathbf{n}_j - \mathbf{n}_i) \cdot \mathbf{l}_x \\ (\mathbf{n}_j - \mathbf{n}_i) \cdot \mathbf{l}_y \end{bmatrix}.$$

Since $\mathbf{n}_i \cdot \mathbf{l}_x = 0$ and $\mathbf{n}_i \cdot \mathbf{l}_y = 0$ we get

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} (\mathbf{v}_j - \mathbf{p}_i) \cdot \mathbf{l}_x \\ (\mathbf{v}_j - \mathbf{p}_i) \cdot \mathbf{l}_y \end{bmatrix} = \begin{bmatrix} \mathbf{n}_j \cdot \mathbf{l}_x \\ \mathbf{n}_j \cdot \mathbf{l}_y \end{bmatrix}. \tag{4.1}$$

We obtain an overdetermined system of linear equations. Using the least squares we find the values a, b and c, which form an estimate of the shape operator. Its eigenvalues are used as estimates of the principal curvatures and stored. We also compute the eigenvector corresponding to the larger eigenvalue and project it back to the tangent plane, yielding one principal direction $\mathbf{e}_1$, which is also stored with the sample.

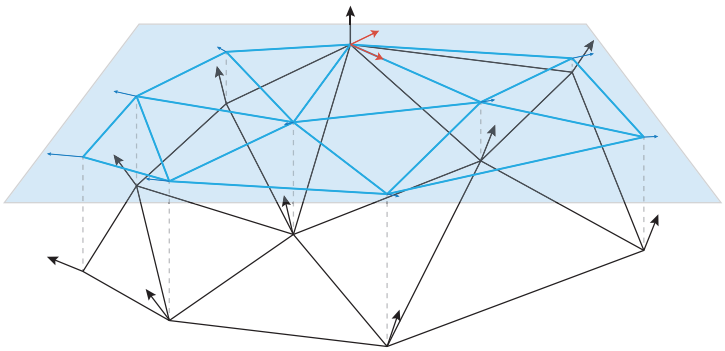


Figure 4.1: Neighborhood for curvature computation. Black lines represent original edges, while blue lines show their projection to the tangential plane, which is parameterized by $(\mathbf{l}_x, \mathbf{l}_y)$, represented by red arrows. Black arrows are 3D normals, while blue arrows are the normals projected to the tangential plane, i.e. $(\mathbf{n}_j \cdot \mathbf{l}_x, \mathbf{n}_j \cdot \mathbf{l}_y)$.

4.1.2 Candidate Rigid Transformations

We sample the vertices from P randomly, using a regular 3D grid over the distribution of points and ensuring that from each cell at most one vertex is selected. This way the sampling is close to uniform, even when the distribution of points in P is not.

Next, we build a set of candidate transformations. We construct a 2D KD tree on the samples from P, using the curvature estimates as coordinates. Next, we sample the vertices of Q using the same strategy as with P, i.e. ensuring a roughly uniform sampling. For each sample point in Q, the principal curvatures are estimated using Eq. (4.1). Having the curvature estimates of a sample from Q, we use the KD-tree to locate the most similar sample from P.

Finally, having a potentially corresponding pair of vertices, we build a pair of rigid transformations T^+ and T^- that map the sample from Q to the sample from P, together with the normal, T^+ maps the principal direction $\mathbf{e}_Q$ of the sample in Q to the principal direction $\mathbf{e}_P$ of the sample in P, while T^- maps $\mathbf{e}_Q$ to $-\mathbf{e}_P$.

For each generated transformation, we quickly verify that it aligns a substantial part of Q. In preprocessing, we build a uniform 3D binary grid over the bounding box of P, in which each cell value determines whether there is a point in P closer to the cell center than δ. We use point sample of Q reduced to about 400 vertices. Each sample vertex $\mathbf{v}_q$ is transformed using the candidate transformation T, and the binary grid is used to determine whether or not there is a point $\mathbf{v}_p$ in P such that $\|T(\mathbf{v}_q) - \mathbf{v}_p\| < \delta$. The percentage of points that meets this criterion is also known as Largest Common Pointset (LCP). If LCP is at least 3%, then the candidate is retained for further processing. We keep sampling Q until a certain number of candidate

transformations is found, which pass the verification.

One possibility to choose the aligning transformation is to pick the candidate with the highest resulting overlap, i.e. the highest portion of samples fulfilling the verification condition. As discussed at the beginning of this chapter, this approach is sensitive to the choice of the δ parameter, and therefore we also use a different approach, exploiting the *distribution of the candidate transformations*.

4.1.3 Analyzing the Space of Rigid Transformations

In the set of candidate transformations, there are probably many transformations quite close to the optimum, but possibly not close enough to obtain a high score. Even in such a situation, however, it is possible to analyze the space of rigid transformations and find a good approximation of the optimum by looking for a density peak in the distribution of the candidate transformations.

The key ingredient needed for evaluating the density is a proper way of evaluating proximity. This problem is challenging, because of the non-trivial topology of the space, which makes its parameterization difficult. Common parameterizations, such as using 4×4 matrices in homogeneous coordinates or dual quaternions, are redundant in the number of parameters used to describe the transformation. The redundancy can be avoided for example by using Euler angles to describe the rotation, however, even then, similarity of two rigid transformations does not imply the similarity of the parameters that describe them. Choosing a proper metric in SE(3) is a crucial task, and therefore we analyze different possibilities in the next section. At this point, we assume that the distances can be consistently measured and finish the overview of our model registration algorithm.

There are different ways of finding a density peak of vertices (rigid transformations in our case) in a vector space. A clustering algorithm can be used that groups similar vectors, and cluster centroids or modes often correspond to density peaks. There are various approaches to clustering, ranging from simple parametric algorithms, such as k-means, up to more complex procedures, such as mean shift clustering [22] or facility location [16]. In each case, the questions remain how to select the cluster which corresponds to the best alignment, how to select the best alignment within the cluster, and whether the clusters correspond to good alignments in the first place. Although some intuitive answers can be found for some of the clustering algorithms, we use a more direct way of finding the density peak, which turns out to be both more efficient computationally, as well as more robust/precise.

A sampling density estimation function can be defined using some kernel function K as

$$\rho(x) = \sum_i K(d(T_i, x)), \tag{4.2}$$

where $d(x, y)$ measures the distance of two samples. Various kernel functions can be used for K, for efficiency reasons we use a simple Gaussian $K(r) = e^{-(Dr)^2}$, where D is a spread parameter.

Instead of looking for the general location of the true global maximum of ρ, we only search for a candidate T^*, for which $\rho(T^*) \geq \rho(T_i)$ for all candidate transformations T_i. Given the spread parameter D and some small threshold t, only samples within a radius $r = \sqrt{-ln(t)}/D$ contribute significantly ($> t$) to the density.

Therefore, the task is to find for each candidate a set of candidates up to a distance r. Such goal is usually achieved using a KD-tree, however, such structure cannot be used in this setting, since the space we are working in is not Euclidean. Having a metric, however, allows using a more general acceleration structure - *the Vantage Point Tree* [103]. It is constructed as a binary tree, where each node represents a certain candidate c. Inner nodes have two child nodes: the *near* node contains all other candidates that are closer to c than a certain threshold, while the *far* node contains candidates that are farther from c than the threshold. The threshold is selected as the median of distances, so that the tree is balanced, and at a certain level the branching is terminated by a leaf node, which contains no more than a certain number of candidates.

Building such structure is of $\mathcal{O}(n \log(n))$ complexity. With the structure built, proximity queries can be resolved quickly with $\mathcal{O}(\log(n))$ complexity in the average case by only investigating the branches which can contribute to the answer. Evaluating the density estimation function at each candidate location has therefore an overall complexity of $\mathcal{O}(n \log(n))$ and it is easily feasible even with the 10^4 candidates used as default setting.

In principle, our approach is similar to what [67] proposed for symmetry detection, however, we use a *substantially different execution* with different metrics in a slightly different application. Our density peak location approach could be also applied in an arbitrary Mode-based method (see Chapter 5) for global symmetry detection. As an alternative, we have tested the k-means algorithm and the local search algorithm which solves the problem in the facility location interpretation. Both algorithms require a higher minimum number of candidates in order to robustly identify the correct alignment than the proposed exhaustive density evaluation. Moreover, k-means is more than $5\times$ slower and the local search for facility location is more than $48\times$ slower. Therefore, we argue that using an advanced clustering algorithm is not an efficient solution for finding the density peak representing the optimal alignment.

4.2 Transformation Distance Metrics

In order to find the density peak in the space of rigid transformations we need to define a metric $d(T_1, T_2)$ which measures the distance of transformation T_1 from transformation T_2. The metric must somehow relate the two parts of the transformation: the rotation and the translation. In this section we therefore discuss other transformation distance metrics that can be used for this purpose. Apart from surface registration, the presented knowledge about rigid transformations can be useful in different areas including symmetry detection, especially in the case of using the Mode-based methods (see Section 5) such as [67] to detect rigid symmetries.

4.2.1 Composed Metrics

Since a rigid transformation can be decomposed into a rotation and a translation, a metric can be defined as $d(T_1, T_2) = c_R d_R(\mathbf{R}_1, \mathbf{R}_2) + c_t\|\mathbf{t}_1 - \mathbf{t}_2\|$ where $d_R(\mathbf{R}_1, \mathbf{R}_2)$ is a metric for rotations, $\mathbf{R}_1$ and $\mathbf{R}_2$ are the rotations of T_1 and T_2 respectively, $\mathbf{t}_1$ and $\mathbf{t}_2$ are translation vectors of T_1 and T_2 respectively and c_R and c_t are customizable coefficients (a constant spread parameter D of the kernel will be used). This way of defining the metric was used e.g. in [50] or [4], a very similar approach was also used by Mitra et al. in [67] with the only difference that the terms are squared and there is an additional term for scaling.

An extensive analysis of several distance metrics for rotations where basically any of these metrics could be used as $d_R(\mathbf{R}_1, \mathbf{R}_2)$ can be found in [43]. Composed transformation distance metrics, as described above, exhibit certain impractical properties, some of which are independent of the choice of $d_R(\mathbf{R}_1, \mathbf{R}_2)$, as discussed next.

Order of Operations

A rigid transformation can be equivalently defined as either a rotation followed by a translation, i.e. $T(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{t}$, or as a translation followed by a rotation, i.e. $T'(\mathbf{x}) = \mathbf{R}'(\mathbf{x} + \mathbf{t}')$. If T and T' describe the same transformation, following must hold:

$$\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{R}'(\mathbf{x} + \mathbf{t}'). \tag{4.3}$$

This equation holds when $\mathbf{R} = \mathbf{R}'$ and $\mathbf{t}' = \mathbf{R}^T\mathbf{t}$, because then $\mathbf{R}'(\mathbf{x} + \mathbf{t}') = \mathbf{R}(\mathbf{x} + \mathbf{R}^T\mathbf{t}) = \mathbf{R}\mathbf{x} + \mathbf{t}$. Applying transformation T to an object corresponds to first changing its orientation to the target orientation by rotation and then translating the object to the target location. Applying T' corresponds to first translating the object to such a position that the subsequent rotation, which enforces the target orientation, also moves the object to the target location.

The change of orientation is the same in both cases. Therefore, Equation 4.3 can be rewritten as follows:

$$\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{R}(\mathbf{x} + \mathbf{t}') \tag{4.4}$$

$$\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{R}\mathbf{x} + \mathbf{R}\mathbf{t}' \tag{4.5}$$

$$\mathbf{t} = \mathbf{R}\mathbf{t}' \Rightarrow \mathbf{R}^T\mathbf{t} = \mathbf{t}' \tag{4.6}$$

Suppose there are two arbitrary rigid transformations T_1 and T_2 expressed in the rotation-first form, i.e. $\mathbf{R}_1\mathbf{x} + \mathbf{t}_1$, $\mathbf{R}_2\mathbf{x} + \mathbf{t}_2$, and translation-first form, i.e. $\mathbf{R}'_1(\mathbf{x} + \mathbf{t}'_1)$, $\mathbf{R}'_2(\mathbf{x} + \mathbf{t}'_2)$. Since $\mathbf{R}_1 = \mathbf{R}'_1$ and $\mathbf{R}_2 = \mathbf{R}'_2$, $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}'_1, \mathbf{R}'_2)$ for any rotation metric d_R. The translation metrics $\|\mathbf{t}_1 - \mathbf{t}_2\|$ and $\|\mathbf{t}'_1 - \mathbf{t}'_2\|$ are, however, different. The first metric can be expanded as shown in Equation 4.7.

$$\|\mathbf{t}_1 - \mathbf{t}_2\| = \sqrt{(\mathbf{t}_1 - \mathbf{t}_2)^T(\mathbf{t}_1 - \mathbf{t}_2)} = \sqrt{\mathbf{t}_1^T\mathbf{t}_1 - 2\mathbf{t}_1^T\mathbf{t}_2 + \mathbf{t}_2^T\mathbf{t}}. \tag{4.7}$$

From Equation 4.6 it follows that $\|\mathbf{t}'_1 - \mathbf{t}'_2\| = \|\mathbf{R}_1^T\mathbf{t}_1 - \mathbf{R}_2^T\mathbf{t}_2\|$ and the second metric can, therefore, be expanded as follows:

$$\begin{aligned} \|\mathbf{t}'_1 - \mathbf{t}'_2\| = \|\mathbf{R}_1^T\mathbf{t}_1 - \mathbf{R}_2^T\mathbf{t}_2\| &= \sqrt{(\mathbf{R}_1^T\mathbf{t}_1 - \mathbf{R}_2^T\mathbf{t}_2)^T(\mathbf{R}_1^T\mathbf{t}_1 - \mathbf{R}_2^T\mathbf{t}_2)} = \\ &= \sqrt{\mathbf{t}_1^T\mathbf{R}_1\mathbf{R}_1^T\mathbf{t}_1 - 2\mathbf{t}_1^T\mathbf{R}_1\mathbf{R}_2^T\mathbf{t}_2 + \mathbf{t}_2^T\mathbf{R}_2\mathbf{R}_2^T\mathbf{t}_2} = \\ &= \sqrt{\mathbf{t}_1^T\mathbf{t}_1 - 2\mathbf{t}_1^T\mathbf{R}_1\mathbf{R}_2^T\mathbf{t}_2 + \mathbf{t}_2^T\mathbf{t}_2}. \end{aligned} \tag{4.8}$$

The only case when $\|\mathbf{t}_1 - \mathbf{t}_2\| = \|\mathbf{t}'_1 - \mathbf{t}'_2\|$ is when $\mathbf{R}_1 = \mathbf{R}_2$. The result of the translation metric and, therefore, of the composed transformation distance metric, depends on the order of rotation and translation, which we choose for describing the transformations. Furthermore, the difference between $\|\mathbf{t}_1 - \mathbf{t}_2\|$ and $\|\mathbf{t}'_1 - \mathbf{t}'_2\|$ grows with the increasing difference between $\mathbf{R}_1$ and $\mathbf{R}_2$.

In principle there is no reason to prefer either representation and its choice seems rather arbitrary. However, for objects located near the origin, the rotation-first form may seem more appropriate, since regardless of the rotation, the direction of the translation vector *approximately* corresponds to the difference between the output and the starting position of the object. Also, the rotation-first form corresponds to the way rigid transformations are evaluated when using matrix multiplication in homogeneous coordinates for transformation evaluation. For these reasons, in the rest of this text, we only consider rigid transformations in the rotation-first form. Nevertheless, the statements made in the rest of this section can be similarly made about the translation-first form using analogous analysis and derivations.

Dependence on Position

Consider an input object Q_{in} and an arbitrary rigid transformation T that transforms Q_{in} into an output object $Q_{out} = T(Q_{in})$. Translating both Q_{in}

and Q_{out} by the same arbitrary vector $\mathbf{t}_0$ results in two objects Q'_{in} and Q'_{out} respectively, with different absolute position but the same mutual position as Q_{in} and Q_{out}. To transform Q'_{in} into Q'_{out} a new transform T' must be defined such that $T'(Q'_{in}) = Q'_{out}$. Since translating Q'_{out} by $-\mathbf{t}_0$ results in Q_{out}, $T(\mathbf{x}) = T'(\mathbf{x} + \mathbf{t}_0) - \mathbf{t}_0$ must hold, which can be expanded as

$$\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{R}'(\mathbf{x} + \mathbf{t}_0) + \mathbf{t}' - \mathbf{t}_0. \tag{4.9}$$

The change of orientation is equal for T and T', and therefore $\mathbf{R} = \mathbf{R}'$, which implies

$$\begin{aligned} \mathbf{R}\mathbf{x} + \mathbf{t} &= \mathbf{R}(\mathbf{x} + \mathbf{t}_0) + \mathbf{t}' - \mathbf{t}_0 \\ \mathbf{t}' &= \mathbf{t} - \mathbf{R}\mathbf{t}_0 + \mathbf{t}_0 \end{aligned} \tag{4.10}$$

Consider two different rigid transformations T_1 and T_2 and two corresponding transformations T'_1 and T'_2, where $\mathbf{t}'_1 = \mathbf{t}_1 - \mathbf{R}_1\mathbf{t}_0 + \mathbf{t}_0$ and $\mathbf{t}'_2 = \mathbf{t}_2 - \mathbf{R}_2\mathbf{t}_0 + \mathbf{t}_0$, according to Equation 4.10, and some arbitrary vector $\mathbf{t}_0$. Since $\mathbf{R}_1 = \mathbf{R}'_1$ and $\mathbf{R}_2 = \mathbf{R}'_2$, it follows that $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}'_1, \mathbf{R}'_2)$, but the translation metrics $\|\mathbf{t}_1 - \mathbf{t}_2\|$ and $\|\mathbf{t}'_1 - \mathbf{t}'_2\|$ are generally different. The first one expands as shown in Equation 4.7, while the second metric expands as

$$\|\mathbf{t}'_1 - \mathbf{t}'_2\| = \|\mathbf{t}_1 - \mathbf{R}_1\mathbf{t}_0 + \mathbf{t}_0 - \mathbf{t}_2 + \mathbf{R}_2\mathbf{t}_0 - \mathbf{t}_0\| \tag{4.11}$$

$$= \|\mathbf{t}_1 - \mathbf{R}_1\mathbf{t}_0 - \mathbf{t}_2 + \mathbf{R}_2\mathbf{t}_0\|, \tag{4.12}$$

and thus $\|\mathbf{t}_1 - \mathbf{t}_2\| = \|\mathbf{t}'_1 - \mathbf{t}'_2\|$ only when $\mathbf{R}_1 = \mathbf{R}_2$ or when $\mathbf{t}_0 = \mathbf{0}$.

Suppose a different scenario, where only the input object Q_{in} is translated by $\mathbf{t}_0$, creating Q'_{in}, and Q_{out} stays the same, i.e. $T(Q_{in}) = T'(Q'_{in}) = Q_{out}$. Now $T(\mathbf{x}) = T'(\mathbf{x} + \mathbf{t}_0)$ must hold, and it holds when $\mathbf{R} = \mathbf{R}'$ and $\mathbf{t}' = \mathbf{t} - \mathbf{R}\mathbf{t}_0$, the derivation is analogical to Eq. 4.10. If we now again consider two transformations T_1, T_2 and corresponding T'_1, T'_2 it is easy to see that $\|\mathbf{t}'_1 - \mathbf{t}'_2\| = \|\mathbf{t}_1 - \mathbf{R}_1\mathbf{t}_0 - \mathbf{t}_2 + \mathbf{R}_2\mathbf{t}_0\|$, which is the same expression as Eq. 4.12.

In a scenario where only Q_{out} is translated by $\mathbf{t}_0$, yielding Q'_{out}, and Q_{in} stays the same ($T(Q_{in}) = Q_{out}$, $T'(Q_{in}) = Q'_{out}$). Now $T(\mathbf{x}) = T'(\mathbf{x}) - \mathbf{t}_0$ must hold and it is easily proven that it holds when $\mathbf{R} = \mathbf{R}'$ and $\mathbf{t}' = \mathbf{t} + \mathbf{t}_0$ because then $\mathbf{R}'\mathbf{x}+\mathbf{t}'-\mathbf{t}_0 = \mathbf{R}\mathbf{x}+\mathbf{t}+\mathbf{t}_0-\mathbf{t}_0 = \mathbf{R}\mathbf{x}+\mathbf{t}$. For two transformations T_1, T_2 and corresponding T'_1, T'_2 we get

$$\|\mathbf{t}'_1 - \mathbf{t}'_2\| = \|\mathbf{t}_1 + \mathbf{t}_0 - \mathbf{t}_2 - \mathbf{t}_0\| = \|\mathbf{t}_1 - \mathbf{t}_2\|. \tag{4.13}$$

This has profound consequences in context of rigid surface registration and possibly in other applications as well including symmetry detection. When the space of rigid transformations is being sampled and transformations are created by fitting points of Q onto points of P, the value of the Euclidean metric applied on the translation components of the transformations

depends on the position of Q (Eq. 4.12), but does not depend on the position of P (Eq. 4.13). Having a general rigid transformation $T_1(\mathbf{x}) = \mathbf{R}_1\mathbf{x}+\mathbf{t}_1$ and $\mathbf{x} = \mathbf{0}$, then $\mathbf{t}_1$ exactly represents the change of position of the point caused by the transformation and if another transformation $T_2(\mathbf{x}) = \mathbf{R}_2\mathbf{x} + \mathbf{t}_2$ is defined then $\|\mathbf{t}_1 - \mathbf{t}_2\|$ exactly represents the difference between the position change of T_1 and the position change of T_2. However, if $\mathbf{x}$ changes by a non-zero vector $\mathbf{t}_0$, then the value $\|\mathbf{t}_1 - \mathbf{t}_2\|$ starts to deviate from the difference of the position changes of T_1 and T_2 and, according to Eq. 4.12, the deviation grows infinitely with the distance of $\mathbf{x}$ from the origin (the length of $\mathbf{t}_0$) and with the difference between the two rotations $\mathbf{R}_1$ and $\mathbf{R}_2$ (the angle between $\mathbf{R}_1\mathbf{t}_0$ and $\mathbf{R}_2\mathbf{t}_0$). This implies that generally the farther a point is from the origin, the worse the translation component $\mathbf{t}$ of a rigid transformation describes the change of the point's position after applying the transformation, and the worse the Euclidean metric of the translation components describes the difference of these changes between the translation components of two arbitrary rigid transformations.

Using the composed transformation metric, which uses the Euclidean metric on the translation components, is therefore only meaningful when the transformed object is approximately centered at the origin. Otherwise the composed metric can behave unpredictably, having a great negative impact on the registration results.

Dependence on Orientation

Consider an input object Q_{in} and an arbitrary rigid transformation T that transforms Q_{in} into an output object $Q_{out} = T(Q_{in})$. Now imagine rotating both Q_{in} and Q_{out} using the same arbitrary rotation matrix $\mathbf{R}_0$ resulting in two objects Q'_{in} and Q'_{out} respectively, with different absolute orientation but the same mutual orientation as Q_{in} and Q_{out}. A transform T' transforms Q'_{in} into Q'_{out}, i.e. $T'(Q'_{in}) = Q'_{out}$. Since rotating Q'_{out} by $\mathbf{R}_0^T$ results in Q_{out}, $T(\mathbf{x}) = \mathbf{R}_0^T T'(\mathbf{R}_0\mathbf{x})$ must hold, which can be expanded as follows:

$$\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{R}_0^T(\mathbf{R}'\mathbf{R}_0\mathbf{x} + \mathbf{t}') \tag{4.14}$$

$$\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{R}_0^T\mathbf{R}'\mathbf{R}_0\mathbf{x} + \mathbf{R}_0^T\mathbf{t}'. \tag{4.15}$$

This equation holds when $\mathbf{R}' = \mathbf{R}_0\mathbf{R}\mathbf{R}_0^T$ and $\mathbf{t}' = \mathbf{R}_0\mathbf{t}$.

Suppose we have two different rigid transformations T_1 and T_2 and two corresponding transformations T'_1 and T'_2, where $\mathbf{t}'_1 = \mathbf{R}_0\mathbf{t}_1$, $\mathbf{t}'_2 = \mathbf{R}_0\mathbf{t}_2$, $\mathbf{R}'_1 = \mathbf{R}_0\mathbf{R}_1\mathbf{R}_0^T$ and $\mathbf{R}'_2 = \mathbf{R}_0\mathbf{R}_2\mathbf{R}_0^T$, and $\mathbf{R}_0$ is some arbitrary rotation matrix. Obviously, $\|\mathbf{t}_1 - \mathbf{t}_2\| = \|\mathbf{t}'_1 - \mathbf{t}'_2\|$ because $\|\mathbf{t}_1 - \mathbf{t}_2\| = \|\mathbf{R}_0\mathbf{t}_1 - \mathbf{R}_0\mathbf{t}_2\|$. In order to satisfy $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}'_1, \mathbf{R}'_2)$ it must be that $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}_0\mathbf{R}_1\mathbf{R}_0^T, \mathbf{R}_0\mathbf{R}_2\mathbf{R}_0^T)$ which holds if the rotation metric d_R is bi-invariant.

Suppose a different scenario, where only the input object Q_{in} is rotated by $\mathbf{R}_0$, creating Q'_{in}, and Q_{out} stays the same, i.e. $T(Q_{in}) = T'(Q'_{in}) = Q_{out}$.

Now $T(\mathbf{x}) = T'(\mathbf{R}_0\mathbf{x})$ must hold, which expands as $\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{R}'\mathbf{R}_0\mathbf{x} + \mathbf{t}'$. This holds for $\mathbf{R}' = \mathbf{R}\mathbf{R}_0^T$ and $\mathbf{t}' = \mathbf{t}$. If we now again consider two transformations T_1, T_2 and corresponding T_1', T_2', obviously $\|\mathbf{t}_1 - \mathbf{t}_2\| = \|\mathbf{t}_1' - \mathbf{t}_2'\|$. For $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}_1', \mathbf{R}_2')$ it must now be that $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}_1\mathbf{R}_0^T, \mathbf{R}_2\mathbf{R}_0^T)$ which holds when the rotation metric d_R is right-invariant.

Finally, when only Q_{out} is rotated by $\mathbf{R}_0$, creating Q'_{out}, and Q_{in} stays the same ($T(Q_{in}) = Q_{out}$, $T'(Q_{in}) = Q'_{out}$), $T(\mathbf{x}) = \mathbf{R}_0^T T'(\mathbf{x})$ must hold, which expands as $\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{R}_0^T(\mathbf{R}'\mathbf{x} + \mathbf{t}')$. This implies $\mathbf{R}' = \mathbf{R}_0\mathbf{R}$ and $\mathbf{t}' = \mathbf{R}_0\mathbf{t}$ because then $\mathbf{R}_0^T(\mathbf{R}'\mathbf{x} + \mathbf{t}') = \mathbf{R}_0^T(\mathbf{R}_0\mathbf{R}\mathbf{x} + \mathbf{R}_0\mathbf{t}) = \mathbf{R}\mathbf{x} + \mathbf{t}$. For two transformations T_1, T_2 and corresponding T_1', T_2', for the translation metric we now again get $\|\mathbf{t}_1 - \mathbf{t}_2\| = \|\mathbf{R}_0\mathbf{t}_1 - \mathbf{R}_0\mathbf{t}_2\|$, which holds. For $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}_1', \mathbf{R}_2')$ we need $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}_0\mathbf{R}_1, \mathbf{R}_0\mathbf{R}_2)$, which holds when the rotation metric d_R is left-invariant.

In context of rigid surface registration, when fitting points of object Q onto points of object P and a composed metric is used to measure distances between the created transformations, the value of the composed metric is independent of the initial orientation of Q only if the rotation metric is right-invariant (i.e. it must hold that $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}_1\mathbf{R}_0, \mathbf{R}_2\mathbf{R}_0)$ for any $\mathbf{R}_0$, $\mathbf{R}_1$, $\mathbf{R}_2$) and independent of the initial orientation of P only if the rotation metric is left-invariant (i.e. $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}_0\mathbf{R}_1, \mathbf{R}_0\mathbf{R}_2)$). For a composed metric to be independent of the initial orientation of both Q and P, the rotation metric must be bi-invariant (i.e. $d_R(\mathbf{R}_1, \mathbf{R}_2) = d_R(\mathbf{R}_1\mathbf{R}_0, \mathbf{R}_2\mathbf{R}_0) = d_R(\mathbf{R}_0\mathbf{R}_1, \mathbf{R}_0\mathbf{R}_2)$), even when the mutual orientation of Q and P remains unchanged. Therefore we recommend only using rotation metrics that are bi-invariant, otherwise the composed metric might behave unpredictably, negatively impacting the registration results.

Rotation Metrics

We consider 6 rotation metrics that are described and analyzed in [43], denoted $\Phi_i, i = 1, ..., 6$, but since $\Phi_6 = 2\Phi_3$, we exclude Φ_6 and only discuss $\Phi_i, i = 1, ..., 5$. For two rotations, described by rotation matrices $\mathbf{R}_1$, $\mathbf{R}_2$, we denote $\mathbf{q}_1$, $\mathbf{q}_2$ the 4-dimensional vectors representing unit quaternions that correspond to the same rotations, and $(\alpha_1, \beta_1, \gamma_1)$, $(\alpha_2, \beta_2, \gamma_2)$ the triplets of corresponding Euler angles.

Distance between Euler angles

$$d_R^{DEA}(\mathbf{R}_1, \mathbf{R}_2) = \Phi_1((\alpha_1, \beta_1, \gamma_1), (\alpha_2, \beta_2, \gamma_2)) =$$

$$= \sqrt{d(\alpha_1, \alpha_2)^2 + d(\beta_1, \beta_2)^2 + d(\gamma_1, \gamma_2)^2}$$

where $d(a, b) = min\{|a-b|, 2\pi-|a-b|\}$ and $\alpha, \gamma \in [-\pi, \pi), \beta \in [-\pi/2, \pi/2)$.

Distance between unit quaternions

$$d_R^{DQ}(\mathbf{R}_1, \mathbf{R}_2) = \Phi_2(\mathbf{q}_1, \mathbf{q}_2) = min\{\|\mathbf{q}_1 - \mathbf{q}_2\|, \|\mathbf{q}_1 + \mathbf{q}_2\|\}$$

where $\mathbf{q}_1$ and $\mathbf{q}_2$ are treated as 4-dimensional vectors and $\|.\|$ denotes the Euclidean norm.

Dot product of unit quaternions

$$d_R^{ADPQ}(\mathbf{R}_1, \mathbf{R}_2) = \Phi_3(\mathbf{q}_1, \mathbf{q}_2) = arccos(|\mathbf{q}_1^T \mathbf{q}_2|)$$

and

$$d_R^{DPQ}(\mathbf{R}_1, \mathbf{R}_2) = \Phi_4(\mathbf{q}_1, \mathbf{q}_2) = 1 - |\mathbf{q}_1^T \mathbf{q}_2|$$

where $\mathbf{q}_1$ and $\mathbf{q}_2$ are treated as 4-dimensional vectors in both cases.

Deviation from identity matrix

$$d_R^{DIM}(\mathbf{R}_1, \mathbf{R}_2) = \Phi_5(\mathbf{R}_1, \mathbf{R}_2) = \|\mathbf{I} - \mathbf{R}_1 \mathbf{R}_2^T\|_F = \|\mathbf{R}_1 - \mathbf{R}_2\|_F$$

where $\|.\|_F$ denotes the Frobenius norm and $\mathbf{I}$ is the identity matrix.

All these metrics, except for d_R^{DEA}, are bi-invariant [43] and therefore well applicable in the context of rigid registration. For more detailed description and further analysis of these rotation metrics see [43]. The proof that d_R^{DEA} is not bi-invariant follows.

Consider general rotation matrices $\mathbf{R}_1$, $\mathbf{R}_2$ and $\mathbf{R}_0$ and the corresponding Euler angles $(\alpha_1, \beta_1, \gamma_1)$, $(\alpha_2, \beta_2, \gamma_2)$ and $(\alpha_0, \beta_0, \gamma_0)$ respectively. If d_R^{DEA} is a bi-invariant rotation metric, then $d_R^{DEA}(\mathbf{R}_1, \mathbf{R}_2) = d_R^{DEA}(\mathbf{R}_1\mathbf{R}_0, \mathbf{R}_2\mathbf{R}_0) = d_R^{DEA}(\mathbf{R}_0\mathbf{R}_1, \mathbf{R}_0\mathbf{R}_2)$ must hold for any $\mathbf{R}_1$, $\mathbf{R}_2$, $\mathbf{R}_0$. Now consider that the rotations are such that $\alpha_1 = 0$, $\beta_1 = 0$, $\gamma_1 = 90°$, $\alpha_2 = 0$, $\beta_2 = -90°$, $\gamma_2 = 0$, $\alpha_0 = 0$, $\beta_0 = 30°$, $\gamma_0 = 60°$. It can be easily shown that the Euler angles of the rotation defined by a matrix $\mathbf{R}_1\mathbf{R}_0$ are $\alpha_{10} = -30°$, $\beta_{10} = 0$, $\gamma_{10} = 150°$ and those of $\mathbf{R}_2\mathbf{R}_0$ are $\alpha_{20} = 0$, $\beta_{20} = -60°$, $\gamma_{20} = 60°$. Similarly, for $\mathbf{R}_0\mathbf{R}_1$ we get $\alpha_{01} = 0$, $\beta_{01} = 30°$, $\gamma_{01} = 150°$ and for $\mathbf{R}_0\mathbf{R}_2$ we get $\alpha_{02} = 73.896°$, $\beta_{02} = -25.658°$, $\gamma_{02} = 56.309°$. It is not hard to show now that $d_R^{DEA}(\mathbf{R}_1, \mathbf{R}_2) = 127.279$, $d_R^{DEA}(\mathbf{R}_1\mathbf{R}_0, \mathbf{R}_2\mathbf{R}_0) = 112.249$ and $d_R^{DEA}(\mathbf{R}_0\mathbf{R}_1, \mathbf{R}_0\mathbf{R}_2) = 119.404$. This implies that in general $d_R^{DEA}(\mathbf{R}_1, \mathbf{R}_2) \neq d_R^{DEA}(\mathbf{R}_1\mathbf{R}_0, \mathbf{R}_2\mathbf{R}_0)$ and $d_R^{DEA}(\mathbf{R}_1, \mathbf{R}_2) \neq d_R^{DEA}(\mathbf{R}_0\mathbf{R}_1, \mathbf{R}_0\mathbf{R}_2)$ thus proving that the rotation metric d_R^{DEA} is neither left nor right-invariant, and therefore not bi-invariant.

Normalization

Since the values of all the rotation metrics described above are bounded on finite intervals, we normalize them so that their values are mapped to $[0, 1]$. We also normalize the translation metric by dividing it by the estimated size of Q which is computed as the average distance of its vertices from its centroid. The composed metric we use can therefore be expressed as $d(T_1, T_2) = c_R \frac{d_R(\mathbf{R}_1, \mathbf{R}_2)}{k_R} + c_t \frac{\|\mathbf{t}_1 - \mathbf{t}_2\|}{k_t}$ where k_R, k_t are the corresponding normalization constants.

Properties of Composed Metrics

In order to use a composed metric in the context of rigid registration, it is advisable to choose a bi-invariant rotation metric and only investigate transformations of a centered object. The centering can also be interpreted as integral part of the metric by evaluating each transformation composed with a centering translation $d(T_1 \circ T_c, T_2 \circ T_c)$, where T_c moves the centroid of the input object to the origin. In this sense, *all composed metrics are data-dependent*, since the position of the centroid of the input object is necessary for their proper evaluation. All composed metrics have two free parameters, c_R and c_t. They influence the scale of the metric and, more importantly, they set the balance of the rotation and translation part.

4.2.2 Compound Metrics

Some of the problems of composed metrics can be eliminated by accepting and exploiting the fact that the metric *inevitably must* depend on the input data to which the transformations are applied, i.e. evaluating the similarity of rigid transformations by evaluating the *difference of their effect* on the input data.

Vertex Sum of Squares

A compound data-induced metric provides a single value which quantifies the difference between the effects of the two transformations to the input object. Such a metric was previously discussed in [79], however, it has only been used in order to evaluate the result of a registration, yet not within the registration process itself. For two rigid transformations T_1, T_2 it is defined as follows:

$$d(T_1, T_2)^2 = \sum_{i=1}^{\nu_q} \|T_1(\mathbf{q}_i) - T_2(\mathbf{q}_i)\|^2 \tag{4.16}$$

where $\mathbf{q}_i$ are the vertices of the input object Q. This is in fact a special case of the Procrustes distance [47] used to measure distances between point clouds with known point-to-point correspondences. In the following, we show that

$$
\begin{aligned}
d(T_1,T_2)^2 = & \sum_{i=1}^{\nu_Q} \|\mathbf{R}_1\mathbf{q}_i + \mathbf{t_1} - \mathbf{R}_2\mathbf{q}_i - \mathbf{t_2}\|^2 = \\
& \sum_{i=1}^{\nu_Q} (\mathbf{R}_1\mathbf{q}_i + \mathbf{t_1} - \mathbf{R}_2\mathbf{q}_i - \mathbf{t_2})^T(\mathbf{R}_1\mathbf{q}_i + \mathbf{t_1} - \mathbf{R}_2\mathbf{q}_i - \mathbf{t_2}) = \\
& \sum_{i=1}^{\nu_Q} (\mathbf{q}_i^T\mathbf{R}_1^T\mathbf{R}_1\mathbf{q}_i + \mathbf{q}_i^T\mathbf{R}_1^T\mathbf{t}_1 - \mathbf{q}_i^T\mathbf{R}_1^T\mathbf{R}_2\mathbf{q}_i - \mathbf{q}_i^T\mathbf{R}_1^T\mathbf{t}_2 + \mathbf{t}_1^T\mathbf{R}_1\mathbf{q}_i + \mathbf{t}_1^T\mathbf{t}_1 \\
& - \mathbf{t}_1^T\mathbf{R}_2\mathbf{q}_i - \mathbf{t}_1^T\mathbf{t}_2 - \mathbf{q}_i^T\mathbf{R}_2^T\mathbf{R}_1\mathbf{q}_i - \mathbf{q}_i^T\mathbf{R}_2^T\mathbf{t}_1 + \mathbf{q}_i^T\mathbf{R}_2^T\mathbf{R}_2\mathbf{q}_i + \mathbf{q}_i^T\mathbf{R}_2\mathbf{t}_2 \\
& - \mathbf{t}_2^T\mathbf{R}_1\mathbf{q}_i - \mathbf{t}_2^T\mathbf{t}_1 + \mathbf{t}_2^T\mathbf{R}_2\mathbf{q}_i + \mathbf{t}_2^T\mathbf{t}_2) \qquad (4.17) \\
d(T_1,T_2)^2 = & 2\sum_{i=1}^{\nu_Q} \mathbf{q}_i^T\mathbf{q}_i + 2(\mathbf{t}_1 - \mathbf{t}_2)^T\mathbf{R}_1 \sum_{i=1}^{\nu_Q} \mathbf{q}_i + 2(\mathbf{t}_2 - \mathbf{t}_1)^T\mathbf{R}_2 \sum_{i=1}^{\nu_Q} \mathbf{q}_i \\
& + \nu_Q\mathbf{t}_1^T\mathbf{t}_1 - 2\nu_Q\mathbf{t}_1^T\mathbf{t}_2 + \nu_Q\mathbf{t}_2^T\mathbf{t}_2 - 2\mathbf{R}_1^T\mathbf{R}_2 : \sum_{i=1}^{\nu_Q} \mathbf{q}_i\mathbf{q}_i^T \qquad (4.18) \\
d(T_1,T_2)^2 = & 2\sum_{i=1}^{\nu_Q} \mathbf{q}_i^T\mathbf{q}_i + \nu_Q\mathbf{t}_1^T\mathbf{t}_1 - 2\nu_Q\mathbf{t}_1^T\mathbf{t}_2 + \nu_Q\mathbf{t}_2^T\mathbf{t}_2 - 2\mathbf{R}_1^T\mathbf{R}_2 : \sum_{i=1}^{\nu_Q} \mathbf{q}_i\mathbf{q}_i^T \qquad (4.19) \\
d(T_1,T_2)^2 = & 2\sum_{i=1}^{\nu_Q} \mathbf{q}_i^T\mathbf{q}_i + \nu_Q\mathbf{t}_1^T\mathbf{t}_1 - 2\nu_Q\mathbf{t}_1^T\mathbf{t}_2 + \nu_Q\mathbf{t}_2^T\mathbf{t}_2 \\
& - 2diag(\mathbf{R}_1^T\mathbf{R}_2) \cdot diag(\sum_{i=1}^{\nu_Q} \mathbf{q}_i\mathbf{q}_i^T) \qquad (4.20)
\end{aligned}
$$

for measuring distances between rigid transformations, it can be evaluated with $\mathcal{O}(1)$ time complexity rather than $\mathcal{O}(n)$.

Expressing T_1 as a rotation $\mathbf{R}_1$ and a translation $\mathbf{t}_1$, and T_2 as a rotation $\mathbf{R}_2$ and a translation $\mathbf{t}_2$, we can derive Eq. (4.17). For any vector $\mathbf{v}$ and any matrix $\mathbf{A}$, it applies that

$$\mathbf{v}^T\mathbf{A}\mathbf{v} = \mathbf{A} : \mathbf{v}\mathbf{v}^T,$$

where $\mathbf{A} : \mathbf{B}$ denotes the Frobenius matrix product. Also

$$\mathbf{v}^T\mathbf{A}\mathbf{v} = \mathbf{A} : \mathbf{v}\mathbf{v}^T = \mathbf{A}^T : \mathbf{v}\mathbf{v}^T = \mathbf{v}^T\mathbf{A}^T\mathbf{v},$$

and thus the expression in Eq. (4.18) can be derived. Without loss of generality, we can shift Q so that $\sum_{i=1}^{\nu_Q} \mathbf{q}_i = \mathbf{0}$, producing the simplified expression in Eq. (4.19).

This approach can be taken one step further: rotate Q so that the matrix $\sum_{i=1}^{\nu_Q} \mathbf{q}_i\mathbf{q}_i^T$ becomes diagonal using EVD. Assuming Q has been rotated this way, we reach the final expression in Eq. (4.20), where $diag(\mathbf{M})$ stands for a vector of diagonal elements of a matrix $\mathbf{M}$. Neither the full matrix $\mathbf{R}_1^T\mathbf{R}_2$ nor $\sum_{i=1}^{\nu_Q} \mathbf{q}_i\mathbf{q}_i^T$ must be computed, only their diagonal elements that contribute to the result.

The final formulation only depends on a scalar value $\sum_{i=1}^{\nu_Q} \mathbf{q}_i^T \mathbf{q}_i$, and a vector $diag(\sum_{i=1}^{\nu_Q} \mathbf{q}_i \mathbf{q}_i^T)$, as has been noted but not exploited in [79]. These quantities can be precomputed for Q, which makes the distance evaluation independent of the size of Q, while still obtaining exactly the same result as when evaluating Eq. (4.16). This dissimilarity measure is fully independent of both rotation and translation and fulfills all properties of a metric.

Triangle Sum of Squares

One disadvantage of the vertex sum of squares is that its value depends on the sampling density of Q, which may be quite irregular. We propose addressing this issue by integrating the squared distance over all triangles rather than summing over vertices. Following the steps of Eq. (4.17-4.20), we derive the value of the integral over a triangle t:

$$d_t(T_1, T_2)^2 = \int_t \|\mathbf{R}_1\mathbf{x} + \mathbf{t}_1 - \mathbf{R}_2\mathbf{x} - \mathbf{t}_2\|^2 da \tag{4.21}$$

$$= 2\int_t \mathbf{x}^T\mathbf{x} da + a(t)(\mathbf{t}_1^T\mathbf{t}_1 - 2\mathbf{t}_1^T\mathbf{t}_2 + \mathbf{t}_2^T\mathbf{t}_2) \tag{4.22}$$

$$- 2diag(\mathbf{R}_1^T\mathbf{R}_2) \cdot diag(\int_t \mathbf{x}\mathbf{x}^T da), \tag{4.23}$$

where $a(t)$ is the area of triangle t. A full rigid transformation metric is obtained by summing over all mesh triangles:

$$d(T_1, T_2)^2 = \sum_{i=1}^{\tau_Q} d_{t_i^Q}(T_1, T_2)^2. \tag{4.24}$$

Note that computationally, the vertex sum and the triangle sum error metrics are equivalent, the only difference is the means of precomputing the constants needed for the evaluation. For a triangle $t = (A, B, C)$, it is not difficult to show that

$$\int_t \mathbf{x}^T\mathbf{x} da = \int_t x^2 da + \int_t y^2 da + \int_t z^2 da, \tag{4.25}$$

$$diag(\int_t \mathbf{x}\mathbf{x}^T dat) = (\int_t x^2 da, \int_t y^2 da, \int_t z^2 da), \tag{4.26}$$

$$\int_t x^2 da = J_t \frac{(A_x + B_x + C_x)C_x + A_x^2 + B_x^2 + A_x B_x}{12}, \tag{4.27}$$

$$J_t = \|(B - A) \times (C - A)\|, \tag{4.28}$$

while the remaining integrals of y^2 and z^2 are expressed analogously. Note that these simplified expressions can only be used for properly centered and pre-rotated mesh, the rotation is equivalent to the vertex sum, only again using integrals over triangles rather than sums. This does not, however, compromise the generality of the approach, since it can be equivalently expressed for general position, only resulting in a more expensive computation by a

constant. Both vertex and triangle sum of squared distances depend on the input data, but only a few values suffice to evaluate the metrics in $\mathcal{O}(1)$ time.

L_1 Data Dependent Metrics

A typical problem of a squared distance error metric is its sensitivity to outliers: much more emphasis is given to larger distances than to small ones. This issue can be alleviated by using a different distance metric [10], such as L_1, leading to:

$$d(T_1, T_2) = \sum_{i=1}^{\nu_Q} \|T_1(\mathbf{q}_i) - T_2(\mathbf{q}_i)\|, \tag{4.29}$$

or if a metric insensitive to irregular sampling is required,

$$d(T_1, T_2) = \sum_{i=1}^{\tau_Q} \int_{t_i^Q} \|T_1(\mathbf{x}) - T_2(\mathbf{x})\| da. \tag{4.30}$$

Notably, a very similar expression has been recently used in order to evaluate the smoothness (similarity) of local transformations needed for non-rigid alignment [55]. With the L_1 norm, it is substantially more difficult to evaluate such metric than when the L_2 norm was used. The vertex sum leads to linear computational complexity, while the triangle sum even leads to integrals that are difficult to evaluate in general. Therefore, instead of the full triangle sum L_1 metric, we propose using an approximation

$$d(T_1, T_2) = \sum_{i=1}^{\tau_Q} a(t_i^Q) \|T_1(\mathbf{c}(t_i^Q)) - T_2(\mathbf{c}(t_i^Q))\|, \tag{4.31}$$

where $\mathbf{c}(t)$ is the centroid of the triangle t. Even with this approximation, it is impractical to use either of the L_1 metrics in registration, because of their computational complexity. Nevertheless, L_1 metrics can be applied for evaluating the quality of the final alignment obtained by registration algorithms when a correct aligning transformation is known.

Alternatively, one could use the well known Hausdorff distance [32] computed between two versions of a mesh transformed by the two input transformations. Such approach, however, ignores the known point-to-point correspondences, and it also cannot be used in registration, because of the computational complexity of the Hausdorff distance evaluation [21].

Normalization

Normalized metrics based on the sum of squares can be written as

$$\frac{1}{r(Q)} \sqrt{\frac{1}{A(Q)} d(T_1, T_2)^2},$$

i.e. the normalization constant is $r(Q)\sqrt{A(Q)}$. The L_1 based metrics are normalized as

$$\frac{1}{r(Q)A(Q)}d(T_1, T_2),$$

where for the vertex-based metrics $A(Q) = \nu_q$ and for the triangle-based metrics $A(Q)$ is the surface area of Q, and $r(Q)$ is the estimated radius of Q. For the vertex-based metrics, $r(Q)$ is computed as the average distance of the vertices of Q from its centroid. For the triangle-based metrics, the average distance is computed for the centroids of all triangles weighted by their areas and the same approach is used for computation of the centroid itself.

If the spread parameter D in the Gaussian kernel for the density peak location is constant, then the compound metrics have a single parameter c, which sets the global scale of the metric. The compound metrics are therefore generally evaluated as $\frac{c}{k}d(T_1, T_2)$ where d is a compound metric and k is the corresponding normalization constant.

4.3 Results

Certain parameters influence the model algorithm, in our experiments, however, we found that the results are not very sensitive to the precise value of any of them. The parameters are:

- Geometric ring of radius 8 times the average edge length has been used as local neighborhood for the curvature estimation.
- 10^4 sample vertices of P were analyzed and their curvatures and principal directions were estimated.
- 10^4 candidate transformations were obtained by sampling Q.
- The distance δ used to determine the LCP of each candidate has been set to 2% of the mesh radius $r(Q)$.
- The minimum value of LCP for a candidate to be accepted has been set to 3%.

The registration algorithm was tested on 14 different realistic registration datasets. Datasets of varying character were chosen, with a range of properties that may cause problems with registration, such as small overlap, noise or potential to registration ambiguity. The reference implementation (current at the time of this research) of the state of the art algorithm Super4PCS was only able to register 7 out of the 14 datasets, and even that was only achieved when the algorithm parameters were adjusted for different

inputs. A detailed report on the performance of Super4PCS on the test data will be presented in Section 4.3.4.

We have tested our model algorithm with the composed metric with all the rotation metrics described in Section 4.2.1 and the two metrics based on sum of squares described in Section 4.2.2. We denote the composed metrics $d^{NAME}(T_1, T_2) = c_R \frac{d_R^{NAME}(\mathbf{R}_1, \mathbf{R}_2)}{k_R} + c_t \frac{\|\mathbf{t}_1 - \mathbf{t}_2\|}{k_t}$ where $NAME$ is the abbreviation for the corresponding rotation metric. Normalized metrics based on vertex and triangle sum of squares are denoted d^{VSS} and d^{TSS} respectively. In all cases the object Q is centered at the origin.

For each of the 14 registration datasets the correct transformation is known and the error of the registration can therefore be measured as the difference between the resulting transformation and the correct one. To compute this error, we use the metric defined by Eq. (4.31), normalized as described in Section 4.2.2, with $c = 1$. We set a threshold ψ, below which we consider a registration successful. In our experiments, correct alignments have had error < 0.1, in order to provide a certain tolerance we set $\psi = 0.15$. We also observed that in case of wrong alignment, the error is usually considerably larger, generally greater than 0.3. Therefore, choosing ψ between 0.1 and 0.3 does not make a big difference.

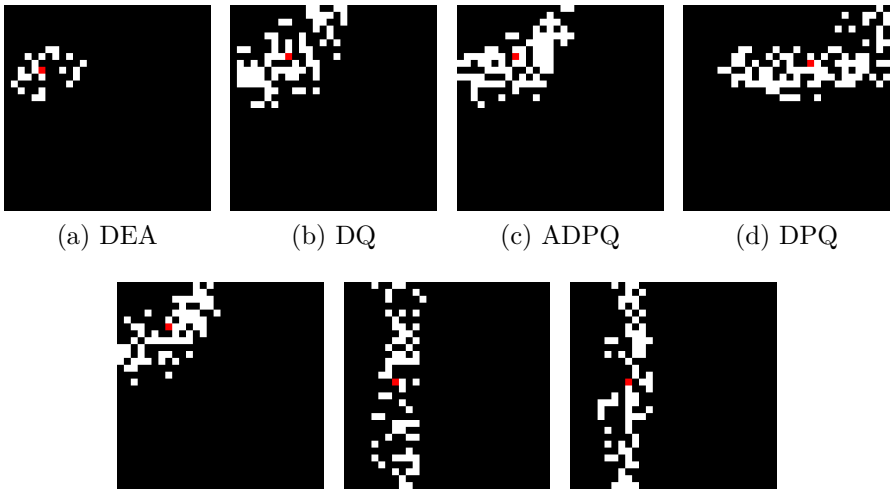
Before using the metrics, their parameters need to be set. In order to do that an experiment was performed, where the spread parameter D in the Gaussian kernel was set to $D = 1$ and 900 different configurations of the c_R, c_t coefficients were created in the following way:

$$c_R = 1.5 \cdot 2^{i/3}, c_t = 1.5 \cdot 2^{j/3}$$

where $i, j = 0, 1, 2, ..., 29$. The exponential progression was used because a good metric should not be very sensitive to the coefficient setting and, therefore, halving or doubling one of the coefficients should not impact the results very much. Since the metrics d^{VSS}, d^{TSS} only depend on a single parameter, we set it as $c = c_R$ while the value of c_t remains unused. With these metrics, the performance does not depend on j.

For each metric, we ran the registration on all the 14 datasets, 5 times at every combination of i and j, which gives 70 tests for every configuration in total. To visualize the results, we created a bitmap of 30×30 pixels, where the index of each pixel corresponds to a single configuration of i and j (i horizontal, j vertical from top to bottom). A pixel is white when the corresponding configuration was successful in all 70 registration cases, i.e. error $< \psi$, otherwise it is black.

For each metric, we computed the centroid of all the white pixels in the bitmap and we used the i, j indices of the centroid pixel to acquire the optimal values of c_R, c_t, which are shown in Table 4.1. The bitmaps are shown in Figure 4.2, the centroid pixels are colored red. Since the algorithm is non-deterministic, even for a good configuration of the coefficients, there



(a) DEA (b) DQ (c) ADPQ (d) DPQ

(e) DIM (f) VSS (g) TSS

Figure 4.2: Bitmaps of the i, j configurations of the coefficient settings for all the metrics. White pixel indicates a successful registration in all 70 test cases, black pixel indicates at least one failure, red pixel shows the centroid of the white pixels and indicates the optimal configuration.

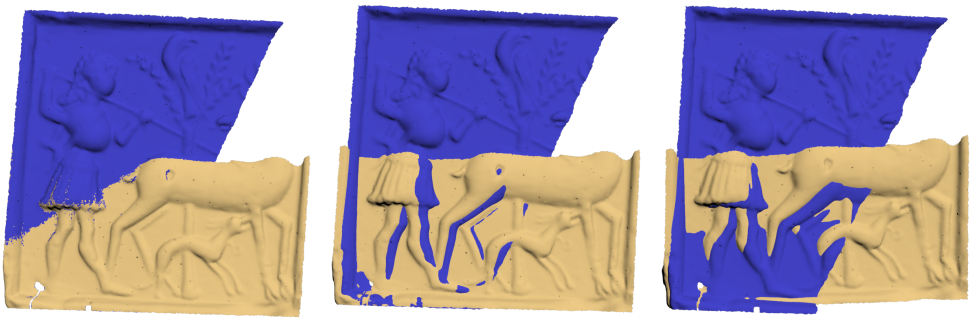
Table 4.1: Optimal coefficients of the transformation distance metrics.

	DEA	DQ	ADPQ	DPQ	DIM	VSS	TSS
c_R	4.76	9.52	9.52	96	7.55	7.55	9.52
c_t	12	7.55	7.55	9.52	6	-	-

is always a certain chance of failure. Therefore, a black pixel does not necessarily mean that the corresponding configuration is bad, especially when it is surrounded mostly by white pixels. On the other hand, if a black pixel lies in a neighbourhood of black pixels, it suggests a configuration far from optimal.

Figure 4.3a shows the registration result of the model algorithm using the d^{TSS} metric with $c = 9.52$ for the *Kac* dataset. The error of this registration was 0.007. For comparison, Figures 4.3b, 4.3c show two artificially created alignments with errors of 0.077 and 0.153 respectively, which are close to $\frac{\psi}{2}$ and ψ.

To perform a comparison of all the metrics, we ran the registration on each of the 14 datasets 1000 times, using the optimal parameters found previously. Table 4.2 shows the average error for each metric on every dataset and the fail count (FC), i.e. how many times of the 1000 tests the registration resulted in error $> \psi$. The error cells are colored according to the order of the error values for the given dataset, i.e. the cell with the largest error is red and the color transits through orange and yellow to green which indicates the cell with the lowest error. The colors do not correspond to the absolute error

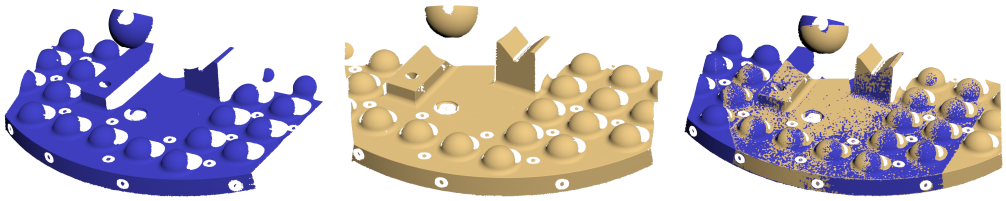


(a) $error = 0.007$ (b) $error = 0.077$ (c) $error = 0.153$

Figure 4.3: Registration result of the model algorithm using the d^{TSS} metric with $c = 9.52$ (a) and its qualitative comparison to two artificial alignments with different errors (b), (c).

Table 4.2: Comparison of the transformation distance metrics. The Error column shows the average errors for corresponding datasets, the FC (fail count) column shows how many times the registration produced an alignment with error $> \psi$.

	DEA		DQ		ADPQ		DPQ		DIM		VSS		TSS	
Dataset	Error	FC	Error	FC	Error	FC	Error	FC	Error	FC	Error	FC	Error	FC
Armadillo	0.0683	23	0.0479	0	0.0485	0	0.0486	0	0.0463	0	**0.0455**	0	0.0480	0
Bird	0.0673	0	0.0672	0	0.0672	0	**0.0669**	0	0.0672	0	0.0673	0	0.0672	0
Bubba	0.0032	0	0.0035	0	0.0039	0	0.0059	0	0.0039	0	0.0034	0	**0.0023**	0
Buddha	0.0363	0	0.0255	0	0.0252	0	0.0246	0	**0.0241**	0	0.0242	0	0.0242	0
Coati	0.0283	0	0.0211	0	0.0208	0	0.0212	0	0.0202	0	**0.0199**	0	0.0200	0
Dragon	0.0258	0	0.0222	0	0.0221	0	0.0233	0	0.0221	0	0.0211	0	**0.0209**	0
Eggs	**0.0758**	98	0.0852	120	0.0868	123	0.0967	142	0.1003	143	0.1202	179	0.0915	140
Head	0.0079	0	0.0081	0	0.0082	0	0.0089	0	0.0083	0	0.0081	0	**0.0075**	0
Hippo	0.0760	32	0.0560	0	0.0558	0	0.0566	0	0.0556	0	**0.0528**	0	0.0538	0
Kachel	0.0187	0	**0.0152**	0	0.0156	0	0.0168	0	0.0153	0	0.0156	0	0.0155	0
Oscar	0.0044	0	0.0039	0	0.0040	0	0.0048	0	0.0039	0	0.0036	0	**0.0034**	0
Suzanne	0.0139	0	0.0133	0	0.0139	0	0.0144	0	**0.0130**	0	0.0132	0	**0.0130**	0
Teeth	0.0160	0	0.0137	0	0.0135	0	0.0148	0	0.0131	0	0.0129	0	**0.0127**	0
Testbody	0.0189	0	0.0175	0	0.0172	0	0.0197	0	0.0181	0	0.0186	0	**0.0171**	0
Total	0.0329	153	0.0286	120	0.0288	123	0.0302	142	0.0294	143	0.0305	179	**0.0284**	140



(a) P (b) Q (c) correct alignment

Figure 4.4: The *Egg* dataset.

values, they only indicate their order for the given dataset. In the rest of this section, we refer to the datasets using a three letter abbreviation. The *Egg* dataset stands out from the others, because it is the only one with non-zero fail count for every metric, suggesting that this model is problematic in some way. The P and Q objects of *Egg* together with their correct alignment are depicted in Figure 4.4. The most likely cause of the failure is the presence of large planar/spherical areas that lead to many incorrect matches, because of the simple curvature based descriptor. Surprisingly, the d^{DEA} metric has the lowest fail count as well as the lowest average error for *Egg*, however, it seems that under the circumstances of a failing descriptor, this result is caused by random influences. Otherwise, d^{DEA} has the largest average error and it is also the only metric with non-zero fail count for two additional datasets: *Arm* and *Hip*. The lowest total average error was achieved with the d^{TSS} metric with d^{DQ} and d^{ADPQ} having slightly larger, but very similar errors. Also, without the problematic *Egg* dataset, the total average errors of the metrics are d^{DEA}: 0.0296, d^{DQ}: 0.0242, d^{ADPQ}: 0.0243, d^{DPQ}: 0.0251, d^{DIM}: 0.0239, d^{VSS}: 0.0236, d^{TSS}: 0.0235, with both the compound metrics d^{VSS} and d^{TSS} having lower error than all the other metrics.

Figure 4.5 shows the ratio of the average error of d^{TSS} (chosen as reference because of its lowest average error) and the average errors of the remaining metrics for all 14 datasets. The figure, together with Table 4.2, shows that there are quite many cases where the d^{DEA} metric shows noticeably larger error than all the remaining metrics (*Arm*, *Bud*, *Coa*, *Dra*, *Hip*, *Kac*, *Tee*). This demonstrates how unreliable a composed metric can be when it uses the non-bi-invariant rotation metric based on distance of Euler angles and, therefore, application of such a metric is not advisable. On several datasets, the d^{DPQ} metric also shows slightly, but noticeably larger error than the other metrics (*Tee*, *Hea*, *Kac*, *Osc*), except for the *Bub* dataset, where the error of d^{DPQ} is much larger in terms of relative comparison.

For the *Bub* dataset, the d^{TSS} metric has the lowest average error and, in relative comparison, even substantially lower than d^{VSS}, even though d^{VSS} and d^{TSS} are based on the same concept. The explanation might be that the *Bub* dataset has quite non-uniformly distributed vertices as shown in Figure 4.6. In such a case, the d^{TSS} metric is expected to work more precisely than

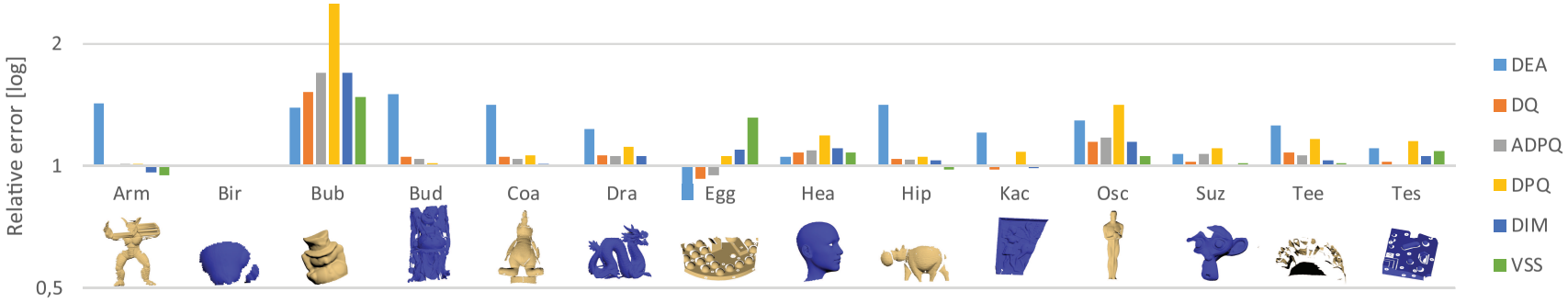


Figure 4.5: Average registration error of each metric normalized by the average error of d^{TSS} for each dataset, shown in log scale.

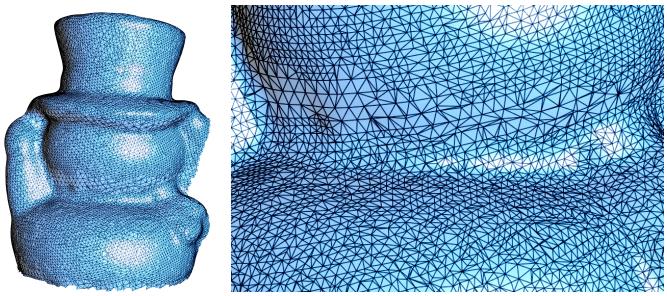


Figure 4.6: The Q object of the *Bub* dataset. The non-uniform distribution of the vertices is quite clear in the detailed view.

d^{VSS}, because it accounts for the triangle size distribution, having more exact information about the object's shape. Similar reason might be behind the lower error of d^{TSS} for other datasets, although there the non-uniform distribution is not so pronounced as with the *Bub* dataset.

4.3.1 Comparing to LCP

We also show results of the registration algorithm where LCP was used instead of the density in the transformation space for evaluating the consensus. The candidate transformation for which the LCP value was the largest was selected as the result of the registration. Such an approach is quite common and was used e.g. by the state of the art registration algorithm Super4PCS [63].

We experimented with the δ parameter of LCP chosen as $\delta = \delta' r(Q)$ where $r(Q)$ is the radius of Q, and we tested several values of δ'. Table 4.3 shows the fail counts of the registration with LCP used for the consensus evaluation with different values of δ' (run 1000 times on each dataset). There is no universal value of δ' and the total fail count is non-negligible for all of the tested values. Although increasing or decreasing δ' leads to decrease of fail count for some datasets, at the same time it causes increase of fail counts for other datasets. This demonstrates that the results of registration algorithms using LCP to select the best candidate transformation depend quite strongly on the choice of δ. On the other hand, when using the density peak location in the space of transformations, although there are some parameters to set (either one or two, depending on the metric), the registration is not very sensitive to their setting, as shown in Figure 4.2 (note that the coefficient growth is *exponential* in both directions).

4.3.2 Non-Centered Object and Density Visualization

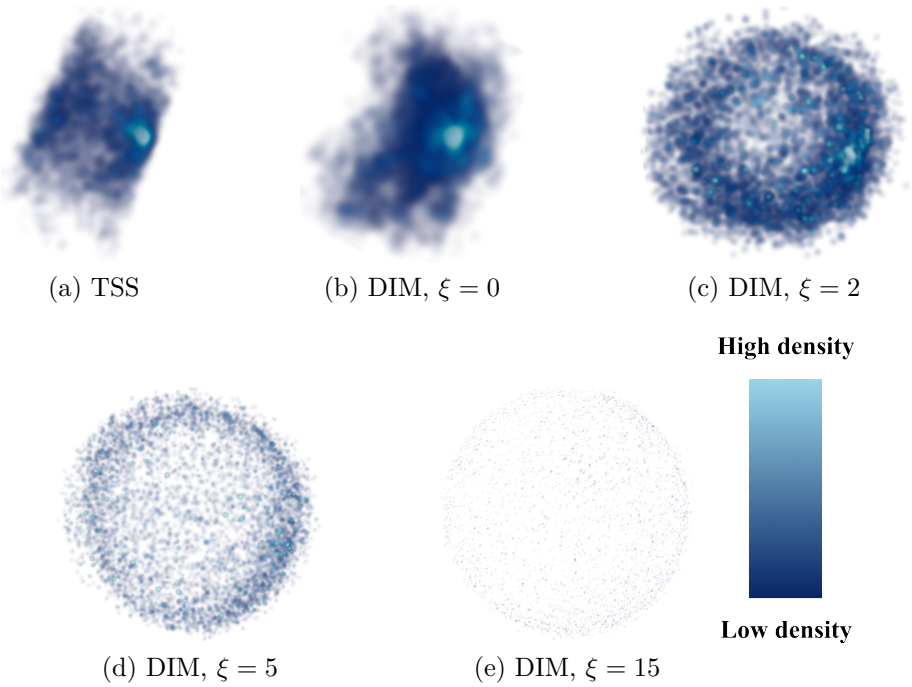
To verify the dependence of the metrics on position as discussed in Section 4.2.1, the registration was executed 1000 times on the *Arm* dataset with the

Table 4.3: Fail counts of the registration with LCP used for evaluation of the consensus with $\delta = \delta' r(Q)$ for different values of δ'.

Dataset	δ'				
	0.0025	0.005	0.01	0.02	0.04
Arm	619	8	0	0	0
Bir	887	757	633	983	1000
Bub	0	0	0	16	1000
Bud	112	0	0	0	0
Coa	59	0	0	0	0
Dra	330	1	0	0	0
Egg	215	4	0	0	0
Hea	0	0	0	0	645
Hip	708	324	219	76	3
Kac	368	0	0	0	327
Osc	0	0	0	0	0
Suz	90	8	6	438	1000
Tee	0	0	0	0	0
Tes	375	55	1	0	0
Total	3763	1157	859	1513	3975

composed metric d^{DIM}, but after centering Q, it was further translated by a vector $\xi \cdot r(Q) \cdot [1, 1, 1]^T$ where ξ determines the final distance of Q from the origin. For $\xi = 0$, the fail count is 0 as in Table 4.2. For $\xi = 2, 5, 15$ and 100, the fail counts were $24, 205, 648$ and 865 respectively. Centering Q is therefore truly crucial for reliable results when using a composed metric and the farther the object is from the origin, the worse results can be expected. In order to visualize the sampling in $SE(3)$ we use a standard multidimensional scaling. First, we compute 3D coordinates that match the candidate distances provided by a metric as closely as possible, obtaining a 3D point cloud. The point cloud is converted to volume data representing the density as defined in Eq. (4.2) and visualized by standard direct volume rendering, choosing an informative viewpoint manually. The visualizations of the decentered dataset registrations are shown in Figure 4.7.

Fig. 4.8 shows another such visualization. It demonstrates that a wrong alignment may produce the highest LCP. On the other hand, a cluster of candidates of rather mediocre LCP is obtained in the vicinity of the correct alignment. More visualizations are also in the accompanying video, showing similar results.



(a) TSS (b) DIM, $\xi = 0$ (c) DIM, $\xi = 2$

(d) DIM, $\xi = 5$ (e) DIM, $\xi = 15$

Figure 4.7: Projection of the candidates into E^3 using MDS for the Arm dataset with TSS and DIM with different values of ξ.

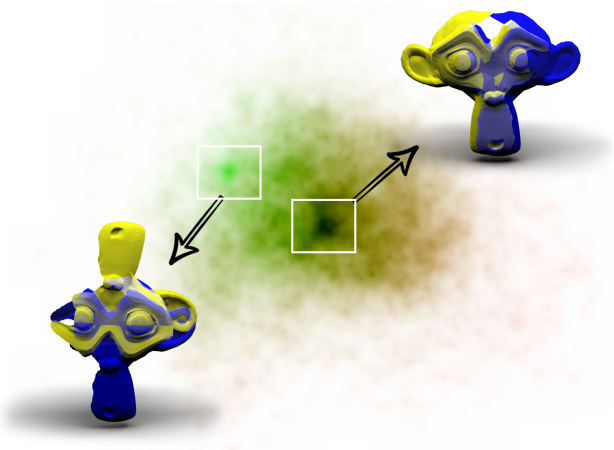


Figure 4.8: 2D visualization of the space of rigid transformation candidates. The hue reflects the score of each candidate, while the brightness reflects the estimated density. The alignments on the left, even though they have the highest score, represent a wrong alignment. A much better alignment on the right can be identified by looking for a density peak.

4.3.3 Noisy Data

The plot in Figure 4.9 shows how the model registration algorithm (using d^{TSS}, $c = 9.52$) behaves on the *Arm* dataset in presence of Gaussian noise with varying standard deviation. The standard deviation is relative to $r(Q)$ and the figure also shows the resulting alignment for deviation of 0.016. The error grows quite slowly up to deviation of 0.018, while for higher values the noise prevents correct registration. Similar results were obtained for other datasets.

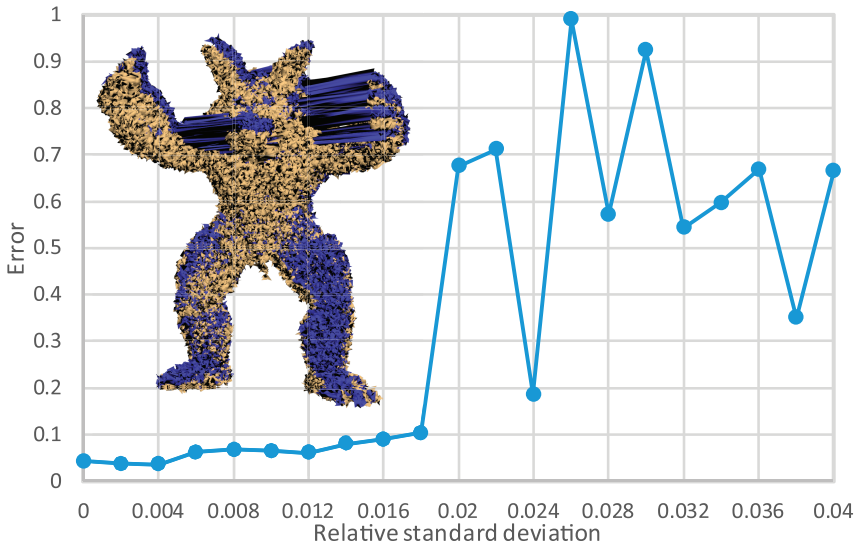


Figure 4.9: Dependence of registration error on Gaussian noise with varying standard deviation relative to $r(Q)$ for the *Arm* dataset. The depicted alignment was achieved with deviation of 0.016.

4.3.4 Comparing to Super4PCS

Table 4.4 shows the errors of the Super4PCS registration method, i.e. of its implementation that was the latest at the time of the research (April 8, 2019). We ran Super4PCS with different values of the δ parameter of its LCP evaluation for all the datasets. The value of delta is set relative to the radius of Q as $\delta = \delta' r(Q)$. The cells with error $\leq \psi$ are marked bold, in all other cases the registration was considered a failure. Names of the datasets for which the registration was succesful for at least one value of δ' are also marked bold, for all the other datasets the registration failed for all the values.

The largest number of successful registrations (7) was achieved using $\delta' = 0.106$, 0.121 and 0.184. Since there is quite a large gap between 0.121

Table 4.4: Errors of Super4PCS with different values of $\delta = \delta' r(Q)$ for all the datasets.

δ'	**Arm**	Bir	Bub	**Bud**	**Coa**	**Dra**	**Egg**	Tee	Hea	**Hip**	Kac	**Osc**	**Tes**	Suz
0.010	1.257	1.941	1.306	1.594	1.387	1.581	1.361	1.416	1.704	1.366	1.801	1.075	1.816	1.366
0.011	0.220	1.941	1.303	1.588	1.383	1.456	1.348	1.396	1.712	1.382	1.834	0.543	1.822	1.376
0.013	1.214	1.941	1.318	1.598	1.387	1.457	1.363	1.380	1.705	1.353	1.826	1.073	1.402	1.341
0.015	1.242	1.941	1.622	1.776	1.245	1.974	1.346	1.383	1.703	1.417	1.841	1.073	1.814	1.596
0.017	**0.099**	1.855	1.296	1.590	1.385	0.418	1.380	1.411	1.710	1.357	1.828	2.016	1.824	1.550
0.020	0.334	0.890	1.464	1.592	1.389	**0.066**	0.166	1.405	1.696	1.048	1.761	1.073	0.391	2.103
0.023	**0.134**	1.319	2.262	0.373	0.225	0.587	1.363	1.398	1.718	1.340	1.852	0.151	1.327	2.099
0.026	0.232	1.577	1.464	0.800	1.380	0.197	1.151	1.397	1.926	1.357	1.844	**0.081**	1.413	1.562
0.030	**0.131**	1.026	1.432	0.273	1.380	2.068	1.295	1.399	1.915	1.096	1.783	**0.067**	0.207	2.039
0.035	**0.080**	1.742	1.454	1.589	1.380	**0.090**	1.568	1.422	1.780	0.443	1.779	**0.146**	0.162	1.752
0.040	0.285	1.277	1.495	0.280	1.378	**0.039**	1.909	1.442	1.671	**0.106**	1.825	**0.135**	**0.122**	1.825
0.046	**0.121**	1.318	1.397	0.209	0.258	**0.060**	0.769	1.443	1.915	**0.094**	1.930	0.161	1.974	1.822
0.053	**0.047**	1.729	1.485	**0.091**	0.484	**0.136**	0.557	1.431	1.964	0.350	1.848	**0.037**	1.275	1.876
0.061	**0.070**	1.411	1.307	**0.064**	**0.120**	**0.065**	1.606	1.414	1.997	0.153	1.721	**0.035**	**0.143**	1.888
0.070	**0.047**	1.421	1.482	0.178	**0.065**	**0.035**	0.207	1.410	1.755	**0.083**	1.593	**0.044**	0.237	1.877
0.080	**0.079**	1.570	1.360	**0.071**	**0.134**	**0.102**	0.383	1.420	1.554	1.685	1.858	**0.079**	**0.105**	1.771
0.092	**0.045**	1.028	1.525	**0.087**	**0.051**	**0.059**	0.519	1.471	1.638	0.168	1.452	**0.105**	1.163	2.003
0.106	**0.071**	1.481	1.548	**0.052**	**0.122**	**0.086**	0.490	1.472	1.946	**0.055**	1.474	**0.074**	**0.040**	2.030
0.121	**0.067**	0.531	1.345	**0.134**	**0.104**	**0.084**	**0.121**	0.467	1.623	**0.117**	1.758	0.165	**0.076**	1.979
0.139	**0.082**	1.558	1.467	**0.075**	**0.164**	**0.126**	0.157	1.558	1.994	0.227	1.237	**0.094**	**0.086**	1.837
0.160	**0.103**	1.473	1.472	**0.062**	**0.048**	**0.069**	0.528	1.503	1.666	0.332	1.639	**0.109**	**0.072**	1.781
0.184	**0.102**	1.410	1.521	**0.073**	**0.071**	**0.128**	0.548	1.585	1.957	**0.140**	1.857	**0.149**	**0.117**	1.723
0.211	**0.119**	1.475	1.505	0.161	**0.141**	**0.136**	0.314	1.500	1.612	0.251	1.639	0.171	**0.148**	1.779
0.243	**0.065**	1.352	1.521	**0.118**	**0.038**	0.178	0.383	0.468	1.947	0.174	1.621	0.260	**0.148**	1.769
0.279	0.162	1.358	1.536	0.152	0.178	0.160	0.401	1.797	1.750	0.201	1.764	**0.147**	**0.076**	1.909
0.320	**0.090**	1.430	1.518	**0.117**	**0.120**	1.228	0.386	1.731	1.907	0.309	1.843	**0.127**	2.002	1.835
0.368	0.256	1.250	1.533	0.183	**0.117**	0.202	0.442	1.936	1.920	0.223	1.898	0.382	0.342	1.767
0.422	1.149	1.431	1.620	1.938	**0.099**	1.412	0.502	2.003	1.895	0.130	1.509	0.603	1.349	1.836
0.485	0.354	1.222	1.622	1.964	0.371	1.610	0.462	2.028	1.785	0.427	1.382	0.244	1.387	1.826
0.557	1.676	1.464	1.628	0.372	0.185	**0.115**	0.676	2.358	2.043	0.391	1.800	0.611	1.385	1.789
0.640	2.128	1.365	1.533	0.446	**0.115**	0.313	0.746	2.422	1.837	0.328	1.405	0.652	1.360	1.621
0.735	2.017	1.497	1.334	0.557	0.259	0.333	1.901	0.329	1.690	0.283	1.463	0.491	1.529	1.748
0.844	1.462	1.278	1.552	1.187	0.426	2.011	1.941	2.626	2.069	1.747	1.428	0.643	1.365	2.560
0.970	1.161	1.284	0.834	0.869	0.303	0.819	1.929	3.507	1.779	1.680	2.208	0.720	1.800	1.514
1.114	2.713	2.683	1.575	1.917	0.554	2.168	0.927	4.015	1.934	1.435	1.944	0.602	1.675	1.688

and 0.184, we select $\delta' = 0.11$ as approximately optimal because it is close to both 0.121 and 0.106. Table 4.5 shows comparison of Super4PCS with $\delta' = 0.11$ to the model RANSAC algorithm where d^{TSS} was used as the metric with $c = 9.52$. For each dataset, ICP alignment was used after the global registration with the same δ setting as Super4PCS ($\delta' = 0.11$). We only show binary evaluation of whether the final registration was successful instead of showing the error, since in the case of successful alignment the final error is dictated by the ICP rather than by the global registration, and in the case of incorrect result the error value is irrelevant. The running times do not include ICP and the measurements were done on a computer with CPU *Intel Core i7-4770* (clock rate 3.4 GHz, 4 cores, L1 cache 256 kB, L2 cache 1 MB, L3 cache 8 MB) and 16 GB of memory with clock rate of 1.6 GHz with *Windows 10 64-bit* operating system.

If the global registration resulted in a good alignment, then the subsequent ICP further strongly increased its precision for most datasets. The

Table 4.5: Comparison of the model RANSAC registration algorithm using d^{TSS} with $c = 9.52$ to Super4PCS with $\delta' = 0.11$, after the global registration ICP was performed also with $\delta' = 0.11$, the running times do not include ICP.

Dataset	Success		Time [ms]	
	Model	S4PCS	Model	S4PCS
Arm	yes	yes	2406	9807
Bir	no	no	820	298
Bub	yes	no	1337	596
Bud	yes	yes	2987	7203
Coa	yes	yes	1958	701
Dra	yes	yes	3071	9084
Egg	yes	no	20654	19592
Hea	yes	no	1369	9693
Hip	yes	yes	1495	1372
Kac	yes	no	1496	635
Osc	yes	yes	5018	4182
Suz	yes	no	6641	5525
Tee	yes	no	2361	1432
Tes	yes	yes	10926	10382

only exception was the *Bir* dataset, where the model registration algorithm actually found a very good alignment, but the ICP made it much worse, resulting in an unsuccessful registration. Super4PCS for this dataset resulted in a bad alignment even without ICP. In total, including ICP did not lead to an improvement of the registration using Super4PCS, which was again successful with 7 datasets. The running times of Super4PCS are comparable to those of our model registration algorithm.

4.3.5 Limitations

Since the model RANSAC algorithm is feature-based, it naturally tends to fail if very strong noise is present in P and Q because in such a case the curvature based features lose reliability. For similar reason, the algorithm does not behave well in case of objects with repetitive patterns or planar areas (e.g. the *Egg* dataset) because there are multiple different spots with similar curvatures, which creates a large potential for registration ambiguity.

The results of the algorithm depend on the size of the overlap of the inputs, both when using the composite and the compound metrics. Either the centering step for composite metrics or the pre-computations for compound metrics may lead to skewed results when the overlap is small, which in turn may negatively influence the registration reliability. This may occur in practice, when for example Q is a result of merging several partial scans,

and has therefore only a small overlap with P. In general, it is therefore advisable to choose as Q the smaller of the two input models, since it has a larger relative portion of overlap.

4.4 Summary

A method of identifying consensus in RANSAC surface registration has been discussed. Instead of searching for a consensus of data points, we look for a consensus of candidate solutions. The consensus is identified with a density peak in the SE(3) space, which is found using the vantage point tree data structure.

In order to locate the density peak, a proper metric is needed in the solution space. We have discussed multiple choices known in the literature and proposed their modified versions. While composed metrics can be applied for this purpose, they suffer from fundamental drawbacks, such as the necessity of setting two free parameters and the lack of translation independence that is merely mitigated by centering the input object. Using the distance of Euler angles as a rotation metric is particularly discouraged, because of its unreliable behaviour.

On the other hand, the equally fast *compound metrics* d^{VSS} and d^{TSS} behave well, regardless of the position of Q, and they exhibit the lowest total average error except for the *Eggs* dataset, where the failure is likely caused by the used feature vector rather than by the metric. Additionally, the proposed d^{TSS} metric is also independent of the sampling density, which makes it the method of choice not only in this, but also in other applications.

Our model registration algorithm turns out to work on par or better than the state of the art algorithm Super4PCS both in terms of speed and reliability. The algorithm is able to identify the best alignment even in situations when the LCP score of a wrong transformation is better than that of the near-optimal alignments. The registration is truly global, fully independent of the initial location of the inputs.

The reliability could be further improved by incorporating more advanced building blocks, such as better local feature vectors or more accurate candidate filtering, since we have chosen rather simple solutions in order to test the consensus evaluation step. As with all global registration methods, a refinement using a variant of ICP is expected to improve the result. The reference implementation of our algorithm is available for download at [36].

The proposed consensus identification strategy could be applied in other registration or symmetry detection algorithms as well, particularly easily in those that can be interpreted in the RANSAC framework, such as Super4PCS. Moreover, it could also be used in other applications, such as dynamic surface segmentation based on motion similarity and others.

We have shown that even the composed metrics are data dependent in

certain sense, relying on the knowledge of the input data centroid. Compound metrics based on the sum of squares retain more information about the input objects in the form of precomputed constants, however, the information is still quite limited. An interesting research direction could be investigating the possibility of formulating metrics that take even more information about the inputs into account, however, without compromising the computational complexity as in the case of L_1 metrics.

Chapter 5

Plane Space Representation in Mode-based Symmetry Plane Detection

Some of the symmetry detection methods described in Chapter 3 can be interpreted as an implementation of one specific and very popular approach It is to create a number of candidate transformations by matching different points or parts of the input object and then find those transformations that occur most often in the transformation space. This can also be described as seeking modes (places of the highest density) in the transformation space so we call this approach Mode-based symmetry detection and it can be classified as a kind of the more general RANSAC approach. For reference, see e.g. the method of Mitra et al. [67], which is most likely the first method that used this approach and one of the most commonly known methods for symmetry detection in general. This approach can be applied in algorithms detecting symmetries of various types, however, in this chapter we only focus on its application for detecting the planes of symmetry (reflectional symmetries) of 3D objects. The content of this chapter was previously published in [40].

The method [67] together with its newer and improved variant [86] are the two more general representatives of the Mode-based approach that can be used to find symmetries of quite a general type. In their case the symmetry transformations can contain rotation, with or without reflection, translation and uniform scaling. The transformation type, however, can be restricted to any subgroup of these general transformations, so these methods can be used to detect reflectional symmetries (symmetry planes) as well. To find the modes both these methods use clustering. Some form of the clustering method [67] was used to detect symmetry planes e.g. in [56], on models of damaged skulls, or in [62] and [48], suggesting its popularity for symmetry plane detection. In Chapter 4 we presented a Mode-based approach used in rigid surface registration. In this case the candidate space contained rigid transformations and to find a single mode, a density peak estimation

algorithm was used. This approach could also possibly be used to find the global plane of symmetry of an object, only with planes as candidates instead of rigid transformations. The Hough transform-based method [13] and the Monte Carlo algorithm used in [77] for 3D surfaces can also be interpreted as Mode-based approaches, only designed to detect specifically symmetry planes. However, instead of finding the mode in a continuous environment, they divide the space of planes into discrete bins to count plane occurrences.

Regardless of the specific algorithm, any Mode-based method for symmetry plane detection requires defining some representation of the space of planes and the result of the method will always somewhat depend on the representation selected. The important aspect of the plane space representation is how well distances between points in the space correspond to the actual similarity/disimilarity of the planes in E^3. In context of symmetry detection, planes can also be understood as transformations reflecting points over the given plane. A useful observation is that the mode(s) can be found in an arbitrary non-Euclidean space only using distances between the points in the space, e.g. as described in [98] or in Section 4.1.3 where we also described how proximity queries in non-Euclidean spaces can be accelerated using the Vantage Point Tree data structure [103]. In Section 4.2 we thoroughly analyzed the problem of computing distances between rigid transformations, however, for reflection transformations or planes the same problem does not seem to be sufficiently addressed in the literature despite the fact that the Mode-based approach is quite popular in the field of symmetry plane detection.

In this chapter we describe and analyze several different representations of the space of planes and for each representation we discuss possible ways of defining a reasonable distance function that could be used in Mode-based symmetry plane detection. We compare these distance functions to a single distance function that we consider the ground truth but which cannot be used in practice because of its large computation cost.

Some of the representations we describe could also be more appropriate for different applications, such as visualization. In general, the information about the plane space representations described below can be useful in any other application, outside the scope of symmetry detection, where some form of plane representation is needed, although the presented distance functions might require some adjustments according to the specific application. We therefore believe that researchers from various fields, not restricted to symmetry plane detection, could benefit from the results of our research.

5.1 Background

Here we provide some necessary background for the rest of this chapter. A general plane P can be defined by the four a, b, c, d coefficients as described

in Section 2.4.3. Here we denote $\mathbf{n} = [a, b, c]^T$ the normal vector of the plane and for the purpose of this chapter we always consider the coefficients to be normalized such that $\|\mathbf{n}\| = 1$ in which case d is the signed distance of the plane from the origin. A function $\mathbf{r}_P(\mathbf{x}) \in E^3$ that reflects an arbitrary point $\mathbf{x} \in E^3$ over the plane P can be defined as shown in Eq. (5.1).

$$\mathbf{r}_P(\mathbf{x}) = \mathbf{x} - 2(\mathbf{n}^T\mathbf{x} + d)\mathbf{n} \tag{5.1}$$

5.1.1 Candidate Creation Algorithm

In order to demonstrate and compare the plane space representations on realistic data we use the following model algorithm for creating planes as candidates for symmetry planes of an arbitrary set of points (point cloud). We use the point set representation because it is more general than a triangle mesh and, therefore, makes the information in this section useful in wider range of applications. Suppose a set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, $\mathbf{x}_i \in E^3, i = 1, 2, ..., N$. We first create a 3D uniform grid with cell size $\frac{l_{avrg}}{\delta} \times \frac{l_{avrg}}{\delta} \times \frac{l_{avrg}}{\delta}$ where l_{avrg} is the estimated size of the object represented by X computed as the average distance of the points of X from their centroid. We mark each cell as either occupied if any point from X falls into it or unoccupied if no point from X falls into it. Then we start randomly selecting pairs of points from X and for each of these randomly selected pairs $\mathbf{x}_i, \mathbf{x}_j$ we create a plane P such that $\mathbf{r}_P(\mathbf{x}_i) = \mathbf{x}_j$. To avoid clutter in the candidate space we perform a quick check to determine whether P is a plausible candidate. This is done by randomly selecting another five points from X, reflecting them over P and checking whether all of them end up in an occupied cell of the previously created grid. If they do P is accepted as a candidate. If at least one of the five points reflects into an unoccupied cell then P is rejected. We keep iterating this process until we have k accepted candidates and if not stated otherwise we set $\delta = 5$ and $k = 2000$. The key idea behind the Mode-based approach is that now there should be significant modes in the candidate space of planes corresponding to the strongest symmetries of the input point set X.

5.1.2 Dependence on Scale and Position

Unlike the a, b, c coefficients, which are bounded on finite interval $\langle -1; 1 \rangle$, the value of the d coefficient of any candidate plane depends on the overall scale and position of the input object represented by the point set X. The dependence on scale is obvious because d represents the distance of the given candidate plane from the origin and if the size of the input object changes, the span of the d coefficient will change as well. However, the a, b, c coefficients will stay the same.

The dependence on position is less obvious. Imagine we translate the input object (all points in X) by some arbitrary vector $\mathbf{t}$, then for an arbitrary

candidate plane P its d coefficient will change by $\mathbf{t}^T\mathbf{n}$ against the value it would have if it was created at the original position. Since the change of d does not depend only on $\mathbf{t}$ but also on the orientation of the given plane, the change of d is inconsistent throughout the candidate planes and this inconsistency is the more significant the farther the input object gets from the origin. For example, suppose a trivial case where an object consists of a single point $\mathbf{x}$ which lies at the origin and suppose a set of all planes passing through $\mathbf{x}$. The d coefficient of all these planes is 0. If we translate $\mathbf{x}$ by $\mathbf{t}$, the d coefficients of the planes that pass through $\mathbf{x}$ will now span from $-\|\mathbf{t}\|$ to $\|\mathbf{t}\|$. Therefore, the position of the input object, i.e. its distance from the origin, again influences the span of d but does not influence the span of a, b, c.

Many of the distance functions for planes, presented later in this text, are negatively influenced by the significantly different span of d and a, b, c. To mitigate this problem, before creating the candidate planes, we always translate the input object so that its centroid is at the origin and, where necessary, we also normalize d by l_{avrg} to make the span of d similar to the span of a, b, c. For those distance functions where the translation to origin is not necessary, this fact will be pointed out explicitly.

5.1.3 Ground Truth

It was already mentioned that in symmetry detection, any candidate plane can be described as reflection transformation as defined by Eq. (5.1). As was pointed out in Chapter 4, distance between transformations cannot be well defined without the context (the object on which the transformations are applied), which is consistent with what was described in Section 5.1.2. Therefore, the distance function for planes that we consider the most meaningful is the point-based version of the metric used for error evaluation of registration results in Chapter 4 (see Eq. (4.29)), only with reflection transformations instead of rigid ones. Given two arbitrary planes P_1 and P_2 the distance function measures the exact difference between the effects of the reflections defined by P_1 and P_2 on the input object. Since such a function gives us the exact evaluation of how differently the input object is effected by the two reflections, we consider it the ground truth distance function, we denote it $D_{GT}(P_1, P_2)$ and it is defined as shown in Eq. (5.2) where $\mathbf{x}_i \in X, i = 1, ..., N$.

$$D_{GT}(P_1, P_2) = \sum_{i=1}^{N} \|\mathbf{r}_{P_1}(\mathbf{x}_i) - \mathbf{r}_{P_2}(\mathbf{x}_i)\| \tag{5.2}$$

The D_{GT} distance function is not effected by the position of the input object, so it does not require the translation to origin, and the object size only effects its overall scale. Unfortunately, the time complexity of computing

D_{GT} is $\mathcal{O}(N)$ where N is the point count of the input object, which makes it too computationally expensive and, therefore, virtually unusable in any Mode-based symmetry detection algorithm. However, we can compare it to the other distance functions described below and measure how close they get to it.

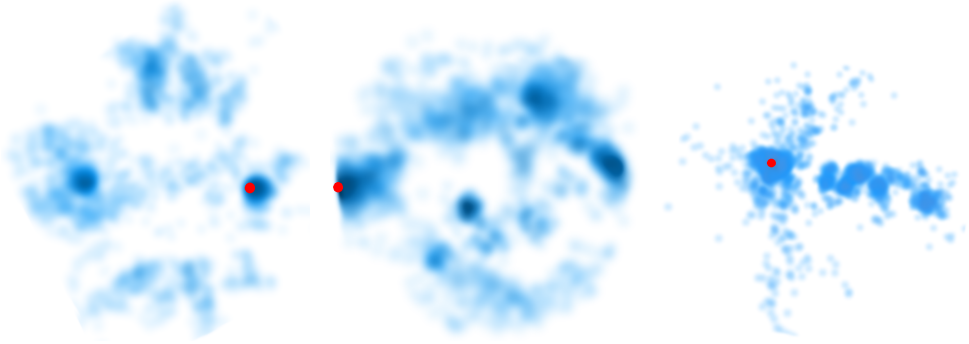
5.2 Plane Space Representations

In this section we describe various different ways of representing the space of planes in E^3 and for each we also describe possible distance functions that can be used in an arbitrary Mode-based symmetry plane detection algorithm. Furthermore, we use the algorithm described in Section 5.1.1 to create a set of candidate symmetry planes of the armadillo object shown in Figure 5.1 and we visualize them in various representations of the plane space. The black line in the figure represents the symmetry plane that we consider the correct one and the object is rotated in such a way that this plane is perpendicular to the plane in which the figure is rendered. We purposely selected an object that is not perfectly symmetrical but still exhibits noticeable reflectional symmetry. Although the object is represented by a triangle mesh, which makes it easier to visualize, only its vertices are used as the points for the candidate creation.

Figure 5.1: Model object with its correct symmetry plane.

5.2.1 Dual Representation in E^3

Although the implicit equation of a plane has four coefficients, there are actually only three degrees of freedom when defining a plane because the space of planes is a 3-dimensional manifold embedded in 4-dimensional space. Therefore, we can use a dual representation of an arbitrary plane as a point in E^3. We denote $\rho(P) \in E^3$ a dual representation of a plane P. Euclidean metric could then be used to compute the distance between two planes P_1, P_2 as $D_\rho(P_1, P_2) = \|\rho(P_1) - \rho(P_2)\|$.



(a) ρ_1 (b) ρ_2 (c) ρ_3

Figure 5.2: Dual representations of the candidate symmetry planes. The colors represent density (the darker, the larger density), the red spot corresponds to the correct symmetry plane.

One possibility to represent a plane in E^3 is to encode its orientation into a vector in E^3 with the same direction as the plane normal vector, and the plane distance from origin into the length of this vector. Such dual representation can be defined as $\rho_1(P) = d\mathbf{n}$. Obviously, for $d \to 0$ such representation gets ambiguous because all planes with $d = 0$ are shrunk into a single point. To solve this problem the value of d can be shifted by a constant μ so that these planes get spread on the surface of a sphere with radius μ instead of being all at the origin. We set $\mu = \frac{1}{2}l_{avrg}$ so that rotating the normal by π and changing d by l_{avrg} make approximately similar change in position of the point in the dual space. The dual representation is therefore finally defined as

$$\rho_1(P) = \begin{cases} (d + \frac{1}{2}l_{avrg})\mathbf{n} & d \geq 0 \\ (d - \frac{1}{2}l_{avrg})\mathbf{n} & d < 0 \end{cases}.$$

Distances in such dual space still do not very well correspond to similarities/disimilarities of the actual planes. Mainly, two planes with d close to 0 and similar normal vectors can be on the other sides of the sphere, and therefore more than 2μ apart, although they are actually very similar. However, such representation can be very good for visualization because each point in the dual space represents the plane quite intuitively.

Figure 5.2a shows the generated candidates on the armadillo model in the dual E^3 space transformed with ρ_1. The darker spots correspond to larger density of the points in the space, the red spot corresponds to the correct plane from Figure 5.1. The viewpoint was selected manually to maximize the information in the image. It can be seen that the correct plane is in a noticeable mode (dense spot) but this mode is split on the surface of the sphere that corresponds to $d = 0$ and its non-negligible part is on the other side. This is quite undesirable because it makes the mode much less significant than it would be if the two parts were together in the space.

Another duality, also called polar duality (described e.g. in [28]), uses normalization of the plane coefficients such that $d = 1$ and then only using the a, b, c coefficients as coordinates in E^3. So instead of multiplying the normal vector by d, this time we divide it by d. This again poses a problem for $d \to 0$ which makes the dual points approach infinity and planes with $d = 0$ cannot be represented at all by this duality. We solve this issue in the same way as with ρ_1 by shifting the d coefficient and we define this dual representation as

$$\rho_2(P) = \begin{cases} \frac{1}{(d+\frac{1}{2}l_{avrg})}\mathbf{n} & d \geq 0 \\ \frac{1}{(d-\frac{1}{2}l_{avrg})}\mathbf{n} & d < 0 \end{cases}.$$

Figure 5.2b shows the candidates transformed by ρ_2 into the dual E^3 space. The correct plane is located in a noticeable mode which is again split into two separate parts that are very far from each other. In this case there are also other significant modes that correspond to very different planes.

Another duality commonly used in computational geometry expresses a plane using its coefficients in explicit representation [7]. There are three possible explicit representations of a plane in E^3:

$$\begin{aligned} x &= -\frac{b}{a}y - \frac{c}{a}z - \frac{d}{a}, \\ y &= -\frac{a}{b}x - \frac{c}{b}z - \frac{d}{b}, \\ z &= -\frac{a}{c}x - \frac{b}{c}y - \frac{d}{c}. \end{aligned}$$

For demonstration, we select the first one, the dual representation is then defined as $\rho_3(P) = [\frac{b}{a}, \frac{c}{a}, \frac{d}{l_{avrg} \cdot a}]$. The division of d by l_{avrg} is necessary for normalizing the span of d. Such duality obviously cannot represent planes parallel to the x-axis and planes with $a \to 0$ approach infinity in the dual space. We could possibly solve this by shifting a but this time, we do not include l_{avrg} into the shift because the span of a does not depend on the size of the input object, so we get

$$\rho_3(P) = \begin{cases} [\frac{b}{a+\frac{1}{2}}, \frac{c}{a+\frac{1}{2}}, \frac{d}{l_{avrg}(a+\frac{1}{2})}] & a \geq 0 \\ [\frac{b}{a-\frac{1}{2}}, \frac{c}{a-\frac{1}{2}}, \frac{d}{l_{avrg}(a-\frac{1}{2})}] & a < 0 \end{cases}.$$

Figure 5.2c shows the candidates in the dual space transformed by ρ_3 and in this case there do not seem to be any significant modes.

In general, the dual representations appear not to be very appropriate for representing planes in any Mode-based symmetry detection algorithm because they all contain singularities. Although this problem can always be solved by shifting the value of some coefficient by a constant, the choice of this constant is rather arbitrary and even then the distances between points

(a) D_{ED} (b) D_{AD} (c) D_{ACD}

Figure 5.3: The candidates represented by 4D vectors projected into E^3 with MDS using different distance functions.

in the dual space might not well correspond to similarities of the planes. However, the dual representations can quite easily be used for visualizing the candidates because the dual points are 3-dimensional.

5.2.2 4D Vector Representation

Probably the most intuitive way of representing a plane is by a 4D vector of the plane coefficients. Given a plane P we represent it by a vector $\mathbf{p} = [a, b, c, \frac{d}{l_{avrg}}]^T$. In such a space we can easily define a distance function as the Euclidean distance of the two 4D vectors. However, $\mathbf{p}$ and $-\mathbf{p}$ represent the same plane so we need to take this into account. The Euclidean distance function is therefore defined as

$$D_{ED}(P_1, P_2) = \begin{cases} \|\mathbf{p}_1 - \mathbf{p}_2\| & \mathbf{p}_1^T\mathbf{p}_2 \geq 0 \\ \|\mathbf{p}_1 + \mathbf{p}_2\| & \mathbf{p}_1^T\mathbf{p}_2 < 0 \end{cases}.$$

In this case the points cannot be visualized directly, so we use the multidimensional scaling (MDS) technique to transform the points into E^3 so that they maintain their distances, w.r.t. the given distance function, as well as possible. However, the projection into E^3 might cause some imprecision in the visualization. Figure 5.3a shows the candidate planes projected into E^3 with MDS using the D_{ED} distance function and there is a very significant mode visible around the correct symmetry plane.

The distances in 4D vector space of planes can also be measured as angles between the vectors because the length of the vector $\mathbf{p}$ does not influence the plane P it represents. The angle distance function can be defined as

$$D_{AD}(P_1, P_2) = arccos\left(\frac{|\mathbf{p}_1^T\mathbf{p}_2|}{\|\mathbf{p}_1\|\|\mathbf{p}_2\|}\right).$$

Figure 5.3b shows the candidates after using MDS with the D_{AD} distance function and the correct plane is again placed inside a noticeable mode.

We can also use only the cosine of the angle and measure its deviation from 1. The angle cosine distance function can be defined as

$$D_{ACD}(P_1, P_2) = 1 - \frac{|\mathbf{p}_1^T \mathbf{p}_2|}{\|\mathbf{p}_1\|\|\mathbf{p}_2\|}$$

and its visualization using MDS is shown in Figure 5.3c. There is again a noticeable mode around the correct plane.

Obviously, the 4D representation of the plane space is much more appropriate for any Mode-based symmetry detection algorithm than the dual representations in E^3. However, they are not as convenient for visualization because in order to show the points they first need to be projected into a lower dimensional Euclidean space which causes a loss of information.

5.2.3 Transformation Representation

As already mentioned, the space of planes can be understood as the space of reflection transformations and, therefore, the distance between arbitrary two planes P_1 and P_2 can be defined as the distance between the two reflection transformations $\mathbf{r}_{P_1}$ and $\mathbf{r}_{P_2}$ defined according to Eq (5.1). One way of doing this is using the compound metric that was evaluated as the most suitable for rigid transformations in Chapter 4. Since in this context we expect a point cloud on the input, not a triangle mesh, we use its point-based version (see eq. (4.16)). It is based on sum of squared distances between the transformed points and for reflection transformations is defined as

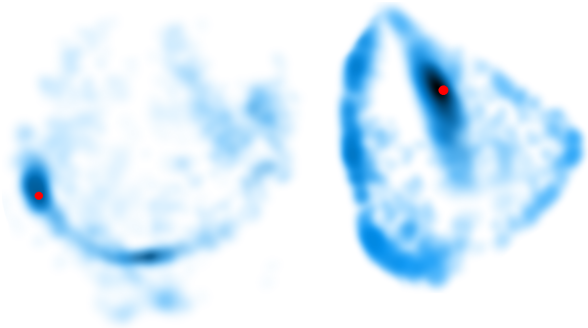
$$D_{SSD}(P_1, P_2) = \sqrt{\sum_{i=1}^{N} \|\mathbf{r}_{P_1}(\mathbf{x}_i) - \mathbf{r}_{P_2}(\mathbf{x}_i)\|^2}$$

where $\mathbf{x}_i \in X, i = 1, ..., N$. There is a notable similarity between D_{SSD} and the ground truth distance function D_{GT} (see Eq. (5.2)). However, as already described in Section 4.2.2 for rigid transformations, there are two major differences. First, D_{SSD} uses squared distances instead of absolute ones, favouring smaller displacements over larger ones, which leads to different distances. Second, unlike D_{GT}, D_{SSD} can be computed in $\mathcal{O}(1)$, given an $\mathcal{O}(N)$ preprocessing is performed beforehand. To achieve this, the transformations must be expressed as $\mathbf{Mx} + \mathbf{t}$ where $\mathbf{M}$ is an orthogonal transformation matrix, $\mathbf{t}$ is an arbitrary translation vector and $\mathbf{x}$ is the transformed point. For the details of deriving the $\mathcal{O}(1)$ computation we refer to Section 4.2.2. Directly from Eq. (5.1) we get

$$\mathbf{r}_P(\mathbf{x}) = \mathbf{x} - 2\mathbf{n}\mathbf{n}^T\mathbf{x} - 2d\mathbf{n} = (\mathbf{I} - 2\mathbf{n}\mathbf{n}^T)\mathbf{x} - 2d\mathbf{n}$$

where $\mathbf{I}$ is the identity matrix. If we now denote $\mathbf{M} = (\mathbf{I} - 2\mathbf{n}\mathbf{n}^T)$ and $\mathbf{t} = -2d\mathbf{n}$, then the reflection transformation can be expressed as

$$\mathbf{r}_P(\mathbf{x}) = \mathbf{Mx} + \mathbf{t}.$$



(a) D_{SSD} (b) D_{DQ}

Figure 5.4: The candidates represented as transformations projected into E^3 with MDS using different distance functions.

Since the matrix $\mathbf{M}$ is orthogonal (and also symmetric), we can use the same derivation as in Section 4.2.2 to compute D_{SSD} in $\mathcal{O}(1)$ with $\mathcal{O}(N)$ preprocessing. The D_{SSD} distance function, as well as D_{GT}, is not effected by the position of the input object, so the initial translation to the origin is not required, and the object size only effects the overall scale of the distance function.

Figure 5.4a shows the candidates projected into E^3 using MDS with the D_{SSD} distance function and the correct plane is again in a significant mode. There is another smaller significant mode visible in the figure, however, this can very likely be caused by the distortion of the MDS projection.

Dual Quaternions

Dual quaternions combine the concepts of quaternions and dual numbers and are commonly used, mainly in robotics, to represent rigid transformations. However, in the following text we show how they can also be used to represent a transformation of reflection over an arbitrary plane. A general quaternion is defined as $Q = q_0 + q_1 i + q_2 j + q_3 k$ where the i, j, k units multiply according to the following rules

$$i^2 = j^2 = k^2 = ijk = -1, ij = k = -ji, jk = i = -kj, ki = j = -ik.$$

Quaternion multiplication is not commutative in general. A conjugate Q^* of a quaternion Q is defined as $Q^* = q_0 - q_1 i - q_2 j - q_3 k$. If $q_0 = 0$ then Q is so called pure quaternion in which case it is that $Q^* = -Q$. For two arbitrary quaternions Q_1 and Q_2 it is that $(Q_1 + Q_2)^* = Q_1^* + Q_2^*$ and $(Q_1 Q_2)^* = Q_2^* Q_1^*$. A size or norm of a general quaternion Q is defined as $\|Q\| = \sqrt{QQ^*}$ which is always a non-negative real number. We denote $v(\mathbf{x}) = xi + yj + zk$ a quaternion that represents an arbitrary point $\mathbf{x} = [x, y, z]^T \in E^3$. If $\mathbf{u}$ is an arbitrary unit vector and we set $Q = cos\alpha + v(\mathbf{u})sin\alpha$, then $Qv(\mathbf{x})Q^*$ represents the point $\mathbf{x}$ rotated by angle 2α around the axis that passes

through the origin and has the direction of $\mathbf{u}$. Also, for any unit vector $\mathbf{u}$ and any α, Q is a unit quaternion, i.e. $QQ^* = Q^*Q = 1$. Similarly, if we set $Q = v(\mathbf{u})$ (as a special case of $\alpha = \frac{\pi}{2}$) then $Qv(\mathbf{x})Q$ represents the point $\mathbf{x}$ reflected over the plane with normal $\mathbf{u}$ that passes through the origin. Notice, that this time Q at the right side is not conjugated. For details about quaternions we refer to [30].

A dual quaternion is defined as

$$Q_d = Q + \epsilon Q_\epsilon = q_0 + q_1 i + q_2 j + q_3 k + \epsilon(q_{\epsilon 0} + q_{\epsilon 1} i + q_{\epsilon 2} j + q_{\epsilon 3} k)$$

where Q and Q_ϵ are quaternions and ϵ is the dual unit which commutes with the quaternion units i, j, k and it is that $\epsilon^2 = 0$. A quaternion conjugate of Q_d is defined as $Q_d^* = Q^* + \epsilon Q_\epsilon^*$, a dual conjugate of Q_d is defined as $\overline{Q_d} = Q - \epsilon Q_\epsilon$. These conjugations can be combined into $\overline{Q_d^*} = Q^* - \epsilon Q_\epsilon^*$.

We denote $v_d(\mathbf{x}) = 1 + \epsilon v(\mathbf{x}) = 1 + \epsilon(xi + yj + zk)$ a dual quaternion that represents an arbitrary point $\mathbf{x} = [x, y, z]^T \in E^3$. If Q is a quaternion that represents rotation and $Q_\epsilon = \frac{v(\mathbf{t})Q}{2}$ where $\mathbf{t} = [t_x, t_y, t_z]^T$ is an arbitrary translation vector then for $Q_d = Q + \epsilon Q_\epsilon$, it can be shown that the expression $Q_d v_d(\mathbf{x})\overline{Q_d^*}$ represents a rigid transformation in the following way. Since $\overline{Q_d^*} = Q^* - \epsilon\frac{(v(\mathbf{t})Q)^*}{2}$ and $(v(\mathbf{t})Q)^* = Q^* v(\mathbf{t})^*$ the expression can be expanded as

$$Q_d v_d(\mathbf{x})\overline{Q_d^*} = (Q + \epsilon\frac{v(\mathbf{t})Q}{2})(1 + \epsilon v(\mathbf{x}))(Q^* - \epsilon\frac{Q^* v(\mathbf{t})^*}{2}).$$

By multiplying the brackets while respecting that $\epsilon^2 = 0$ we get

$$Q_d v_d(\mathbf{x})\overline{Q_d^*} = QQ^* - \epsilon\frac{QQ^* v(\mathbf{t})^*}{2} + \epsilon Q v(\mathbf{x}) Q^* + \epsilon\frac{v(\mathbf{t})QQ^*}{2}$$

and since $QQ^* = 1$ we further get

$$Q_d v_d(\mathbf{x})\overline{Q_d^*} = 1 - \epsilon\frac{v(\mathbf{t})^*}{2} + \epsilon Q v(\mathbf{x}) Q^* + \epsilon\frac{v(\mathbf{t})}{2}.$$

Because $v(\mathbf{t})$ is a pure quaternion and therefore $v(\mathbf{t})^* = -v(\mathbf{t})$, the expression finally yields the form

$$Q_d v_d(\mathbf{x})\overline{Q_d^*} = 1 + \epsilon Q v(\mathbf{x}) Q^* + \epsilon v(\mathbf{t}) = 1 + \epsilon(Q v(\mathbf{x}) Q^* + v(\mathbf{t}))$$

which represents the point $\mathbf{x}$ rotated using the quaternion Q and then translated by $\mathbf{t}$. This shows how dual quaternions can be used for representing and computing rigid transformations. Note that Q_d represents the same transformation as $-Q_d$ with the identity being represented by either 1 or -1. Also, the transformations can be concatenated by multiplying the corresponding dual quaternions and if Q_d represents a rigid transformation then Q_d^* represents its inverse. This in turn means that given two dual quaternions Q_{d1}, Q_{d2} representing rigid transformations, these transformations are the same only

if $Q_{d1}Q_{d2}^* = 1$ or $Q_{d1}Q_{d2}^* = -1$. For details about dual quaternions we refer to [81] or [76].

Consider now a plane P and a dual quaternion $Q_d = Q + \epsilon Q_\epsilon$ defined such that $Q = v(\mathbf{n})$ and $Q_\epsilon = \frac{v(\mathbf{t})Q}{2}$ where $\mathbf{t} = -2d\mathbf{n}$. Now Q_d represents a transformation that first rotates by π around the axis that passes through the origin and has the direction of $\mathbf{n}$, and then translates by $-2d\mathbf{n}$. However, if we apply the transformation on $-\mathbf{x}$ instead of $\mathbf{x}$, it can be easily shown that the transformation expression expands as

$$Q_d v_d(-\mathbf{x})\overline{Q_d^*} = 1 - \epsilon Q v(\mathbf{x}) Q^* + \epsilon v(\mathbf{t}).$$

Because $Q = v(\mathbf{n})$ is a pure quaternion, hence $Q^* = -Q$, we can adjust the expression as

$$Q_d v_d(-\mathbf{x})\overline{Q_d^*} = 1 + \epsilon Q v(\mathbf{x}) Q + \epsilon v(\mathbf{t}) = 1 + \epsilon(Q v(\mathbf{x}) Q + v(\mathbf{t})) =$$

$$= 1 + \epsilon(v(\mathbf{n})v(\mathbf{x})v(\mathbf{n}) - v(2d\mathbf{n})) = v_d(\mathbf{r}_P(\mathbf{x}))$$

which exactly represents $\mathbf{r}_P(\mathbf{x})$. This shows that a dual quaternion can also represent a reflection transformation by representing a rigid transformation that transforms $-\mathbf{x}$ to $\mathbf{r}_P(\mathbf{x})$. Therefore, to measure distances between reflection transformations we can use a distance function for dual quaternions.

We denote $vec(Q_d) = [q_0, q_1, q_2, q_3, q_{\epsilon 0}, q_{\epsilon 1}, q_{\epsilon 2}, q_{\epsilon 3}]^T \in E^8$ an 8-dimensional vector that is equivalent to Q_d. Given a plane P, we create the corresponding dual quaternion Q_d such that $Q = v(\mathbf{n})$ and $Q_\epsilon = \frac{v(-2d\mathbf{n})Q}{2l_{avrg}}$, i.e. $Q_d = v(\mathbf{n}) + \epsilon\frac{v(-2d\mathbf{n})v(\mathbf{n})}{2l_{avrg}}$. The division by l_{avrg} is again to normalize the d coefficient (which of course cannot be done if the dual quaternion is intended to be used for computing the reflection transformation). Since $v(-2d\mathbf{n}) = -2d\, v(\mathbf{n})$ and because $v(\mathbf{n})$ is a pure unit quaternion we can get that

$$Q_\epsilon = \frac{-2d\, v(\mathbf{n})v(\mathbf{n})}{2l_{avrg}} = \frac{d\, v(\mathbf{n})^* v(\mathbf{n})}{l_{avrg}} = \frac{d}{l_{avrg}}$$

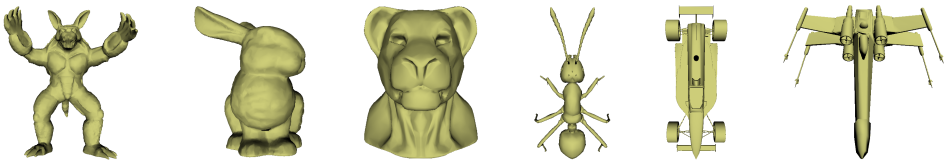
so Q_d can be finally expressed as shown in Eq. (5.3).

$$Q_d = v(\mathbf{n}) + \epsilon\frac{d}{l_{avrg}} = ai + bj + ck + \epsilon\frac{d}{l_{avrg}} \tag{5.3}$$

Notice that the transition from the implicit equation of a plane to the corresponding dual quaternion is very straightforward because the a, b, c, d coefficients of the plane exactly correspond to four of the eight values of the dual quaternion and the remaining four values are 0.

There are two common distance functions for dual quaternions. The first one uses differences between the equivalent 8-dimensional vectors [76]. Suppose two arbitrary planes P_1 and P_2 represented by dual quaternions Q_{d1} and Q_{d2} respectively. Such distance function can be defined as

$$min\{\|vec(Q_{d1}) - vec(Q_{d2})\|, \|vec(Q_{d1}) + vec(Q_{d2})\|\}$$



(a) Armadillo 10026 (b) Bunny 9831 (c) Lion 2213 (d) Ant 3495 (e) Formula 10969 (f) Space ship 3099

Figure 5.5: The test objects used to generate the candidate sets for comparing the distance functions. The number under the name of each object expresses its point count.

but given Eq. (5.3) this is exactly the same as D_{ED}. The second distance function [27] uses a difference transformation $Q_{d1}Q_{d2}^*$ and computes its distance from the identity, i.e. from 1 or -1. It is defined as

$$D_{DQ}(P_1, P_2) = min\{\|vec(1 - Q_{d1}Q_{d2}^*)\|, \|vec(1 + Q_{d1}Q_{d2}^*)\|\}.$$

Figure 5.4b shows the candidates projected into E^3 using MDS with D_{DQ} and the correct plane is in an obvious mode.

5.3 Results

We compared the distance functions by generating the candidate symmetry planes of a given object (using the model algorithm described in Section 5.1.1), computing distances between them using the given distance function and comparing them to distances computed using the ground truth distance function. We did this for the six different test objects shown in Figure 5.5, taken from datasets [51] [26]. The objects are represented by triangle meshes for easier visualization, but we again only used their vertices as the input points for the candidate creation process. The Armadillo and Bunny objects are simplified because the computation of D_{GT} on their original version would be too timely for the experiments we performed.

Let $C = \{P_1, P_2, ..., P_k\}, k = 2000$ be the set of candidate planes created for a given input object. The error of a given distance function D against the ground truth is defined as

$$Err(D) = \frac{1}{Count(k)} \sum_{i=1}^{k} \sum_{j=i+1}^{k} \left| \frac{D_{GT}(P_i, P_j)}{Avrg(D_{GT})} - \frac{D(P_i, P_j)}{Avrg(D)} \right|$$

where

$$Avrg(D) = \frac{1}{Count(k)} \sum_{i=1}^{k} \sum_{j=i+1}^{k} D(P_i, P_j)$$

Table 5.1: Errors of the distance functions for the candidate sets for different objects.

	Arm	Bun	Ant	For	Lio	Shi	Average
D_{ED}	0.120	0.277	0.163	0.093	0.130	0.234	0.169
D_{AD}	0.133	0.281	0.157	0.098	0.144	0.236	0.174
D_{ACD}	0.299	0.388	0.264	0.250	0.306	0.352	0.309
D_{SSD}	0.012	0.023	0.009	0.014	0.011	0.012	0.013
D_{DQ}	0.118	0.277	0.162	0.093	0.129	0.232	0.168
D_{ρ_1}	0.382	0.399	0.503	0.596	0.326	0.425	0.438
D_{ρ_2}	0.401	0.408	0.488	0.563	0.360	0.489	0.451
D_{ρ_3}	0.280	0.446	0.269	0.730	0.362	0.447	0.422

is the average distance between candidates in C and $Count(k) = \frac{1}{2}(k^2 - k)$ is the total number of candidate pairs used for the computation. The normalization by $Avrg$ is used because the overall scales of the distance functions do not matter so the differences are computed after both D_{GT} and D are divided by their mean values.

Table 5.1 shows the errors of all the distance functions described above for all the test objects. For completeness, we include the dual representations in the comparison.

The smallest error is obviously achieved using D_{SSD} which is probably due to D_{SSD} and D_{GT} being based on the same principal. However, it is still rather surprising that the D_{SSD} function which uses squared distances is so similar to D_{GT} that uses absolute distances. The D_{ED}, D_{AD} and D_{QD} all exhibit very similar errors (with D_{QD} usually having the lowest of these three) which are overall lower than those of D_{ACD} and the distances in the dual spaces, but in case of D_{ACD} this can be explained by its resemblance to the cosine function ($D_{ACD} = 1 - cos(D_{AD})$). The function D_{ρ_3} exhibits similar or lower error than D_{ACD} on some objects (Arm, Ant) but also considerably larger error on different ones (For, Shi) which suggests that D_{ρ_3} is quite unpredictable.

The graphs in Figures 5.6, 5.7, 5.8 show the relation between D_{GT} and the other distance functions. We generated 50 candidates on the Armadillo object and for each pair of the candidates we put its distance computed using D_{GT} on the horizontal axis and the distance computed using a given different distance function on the vertical axis. We divide each value by the mean of the given distance function computed by $Avrg$. If some distance function D was exactly the same as D_{GT} (apart from overall scale) then there would be a perfect linear dependency and the points for D would lie on a perfect line in the graph. Figure 5.6 shows the relations of D_{SSD}, D_{ED} and D_{ACD} to D_{GT}. The functions D_{AD} and D_{DQ} are not included in this figure because they are too similar to D_{ED}, this similarity is shown separately in Figure

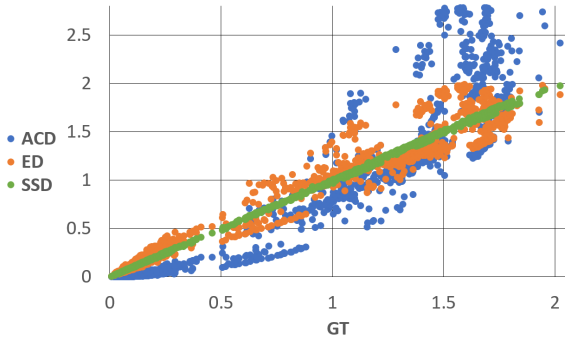


Figure 5.6: Relations between $D_{SSD}/D_{ED}/D_{ACD}$ and D_{GT} for the Armadillo object.

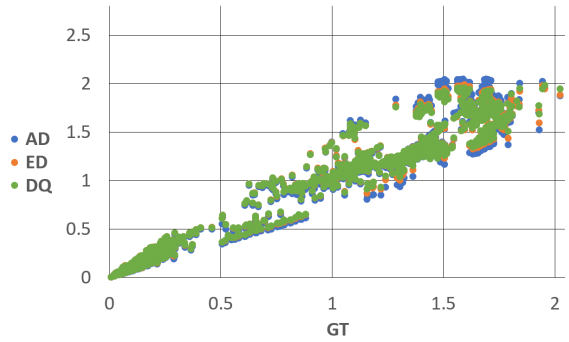


Figure 5.7: Relations between $D_{DQ}/D_{ED}/D_{AD}$ and D_{GT} for the Armadillo object.

5.7. The dual representations are also shown separately in Figure 5.8. For different objects the graphs are slightly different but overall very similar. There is an obvious almost linear dependency between D_{GT} and D_{SSD} (see Fig. 5.6), however, D_{ED}, D_{AD} and D_{QD} exhibit relation to D_{GT} that is also quite near linear dependency (see Fig. 5.7). For D_{ACD} the resemblance to cosine is visible in the graph. On the other hand, the dual representations show rather unstable behavior (see Fig. 5.8). This is mostly caused by the shift in some of the coordinates in the dual space but without this shift all the dual representations would suffer from singularities which is even worse and makes them virtually unusable. It is possible that shifting by a different constant could lead to better results, at least for some objects, however, the choice of the shifting constant is arbitrary and there does not seem to be any reasonable way to set it appropriately.

Table 5.2 shows the Pearson correlations [68] between all pairs of the distance functions (including D_{GT}) for the data shown in Figures 5.6, 5.7, 5.8. Value of 1 indicates perfect linear dependency and the closer the value is to 0 the weaker the linear dependency is. Expectedly, D_{SSD} shows the best linear correlation with D_{GT}. However, the correlations of D_{ED}, D_{AD}, D_{DQ}

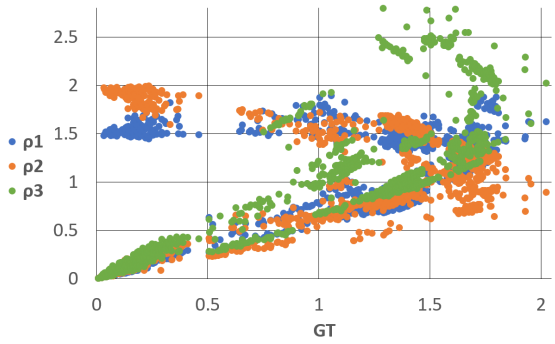


Figure 5.8: Relations between $D_{\rho_1}/D_{\rho_2}/D_{\rho_3}$ and D_{GT} for the Armadillo object.

Table 5.2: Pearson correlations of the distance functions for the Armadillo object.

	GT	ED	AD	ACD	SSD	DQ	ρ_1	ρ_2	ρ_3
GT	1.0000	0.9723	0.9644	0.9120	0.9998	0.9738	0.5361	0.3265	0.7238
ED	0.9723	1.0000	0.9989	0.9679	0.9713	0.9998	0.5537	0.3460	0.7621
AD	0.9644	0.9989	1.0000	0.9767	0.9635	0.9983	0.5499	0.3474	0.7548
ACD	0.9120	0.9679	0.9767	1.0000	0.9105	0.9664	0.5196	0.3111	0.7262
SSD	0.9998	0.9713	0.9635	0.9105	1.0000	0.9728	0.5351	0.3286	0.7214
DQ	0.9738	0.9998	0.9983	0.9664	0.9728	1.0000	0.5516	0.3423	0.7637
ρ_1	0.5361	0.5537	0.5499	0.5196	0.5351	0.5516	1.0000	0.9248	0.4217
ρ_2	0.3265	0.3460	0.3474	0.3111	0.3286	0.3423	0.9248	1.0000	0.1934
ρ_3	0.7238	0.7621	0.7548	0.7262	0.7214	0.7637	0.4217	0.1934	1.0000

and even D_{ACD} with D_{GT} are all rather high, all above 0.9. On the other hand, the distances in the dual spaces exhibit mostly low correlation with D_{GT}. Notably, the correlations among D_{ED}, D_{AD} and D_{DQ} are all very high, confirming that these three distance functions are indeed all very similar.

5.3.1 Theoretical Comparison

Based on the results, the most appropriate representation of the space of planes in any Mode-based symmetry detection method is the transformation representation with the D_{SSD} distance function. But the results also suggest that, except for the dual representations, all the distance functions are rather similar and none of them deviates significantly from D_{GT} which makes all of them well applicable. One only has to keep in mind that all the distance functions except D_{SSD} require translating the input object to the origin, otherwise the normalization of the d coefficient would have to be done differently. But if the practical results are put aside, there are yet some theoretical differences between the various representations.

Since the dual and the 4D vector representations are basically Euclidean, using these representations the candidates can easily be stored in some Euclidean data structure, such as a KD-tree or a grid, that can be used for fast proximity queries if needed. In case of the D_{AD} and D_{ACD} distance

functions some structure can possibly be built using the polar coordinates in 4D. Also, there are quite many possible algorithms for mode-seeking in Euclidean data. The transformation representations and the D_{SSD} and D_{DQ} distance functions are non-Euclidean and therefore slightly more restrictive in terms of the accelerating data structures and possible mode seeking algorithms that can be used. However, as already pointed out, such possibilities exist [98][103] (see also Section 4.1.3). Also, the implementation of the D_{SSD} and D_{DQ} is more complex since D_{SSD} requires some matrix representation and basic operations of matrix algebra and D_{DQ} requires implementing the operations of dual quaternion algebra. On the other hand, the dual and 4D vector representations only require the ordinary algebraic operations with real numbers and vectors.

We also note that, although the D_{DQ} distance function does not seem to bring any considerable improvement over the other simpler distance functions, the idea of representing reflections by dual quaternions seems novel and can possibly find its use in different applications or if some better distance functions for dual quaternions occur in the future. It could also be used when creating new symmetry detection algorithms to represent the planes, possibly taking advantage of the dual quaternion algebra.

5.4 Summary

We have described several representations of the space of planes that can be used in any Mode-based algorithm for symmetry plane detection and we have also described how distances can be computed in the various space representations. We have shown that the 3-dimensional dual space representations are not very appropriate for this purpose but they can easily be used for visualization purposes. In order to represent the space of planes appropriately in the Mode-based symmetry detection, spaces of higher dimensionality need to be used and the transformation representation, which appears to be the most appropriate one, is even non-Euclidean. However, the results suggest that apart from the 3D dual spaces all the plane space representations are well applicable in this context, although there are some theoretical differences between them.

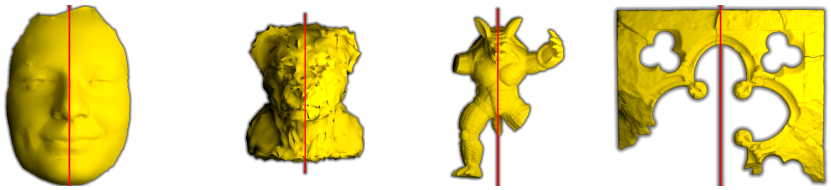
Chapter 6

Symmetry Plane Detection Using Differentiable Symmetry Measure

In the previous chapter we analyzed different ways of representing planes in methods for symmetry plane detection that are based on the popular Mode-based approach. In this chapter, we describe a new method for detecting the global symmetry plane of a 3D object which is based on a completely different approach. The content of this chapter was previously published in [42].

Since reflectional symmetry is probably the most often occurring symmetry in real world objects, having a reliable and robust method for symmetry plane detection in 3D data can be very useful for many applications. In general, it is desirable for the method to be capable of detecting symmetries in objects that are only approximately symmetrical (see e.g. Fig. 6.1a) or symmetries in objects damaged by noise (see Fig. 6.1b). Some applications, such as object reconstruction [97, 91, 85], even require the symmetry detection to work on objects where some parts are missing, which strongly disrupts the symmetry and makes it much more difficult to detect. See, for example, the objects in Figures 6.1c and 6.1d. Although the global reflectional symmetry in them is quite weak, a human observer is still able to see it and a good symmetry detection method should be able to find it.

In the following, we propose a surprisingly simple, differentiable symmetry measure, usable for evaluating symmetry of a 3D object, and we use it to design a robust and flexible global symmetry plane detection method. The proposed method can be used on perfectly as well as approximately symmetrical objects and we show that it is also capable of detecting the plane of symmetry for objects with extensive missing parts, as well as for noisy objects. Furthermore, the proposed method takes a general discrete set of points on the input and therefore puts virtually no constraints on the input data, which can be very useful because a more advanced object



(a) Real human face (b) Noisy Lion (c) Partial Armadillo (d) Embrasure

Figure 6.1: An approximately symmetrical object [15] - (a) a noisy object - (b) and two objects with missing parts - (c), (d).

representation, such as a manifold mesh, is not always available. Many previous approaches use some form of descriptors (e.g. [67, 13, 86, 91]), mostly cuvature-based, i.e. the quality of their results depends on the quality of the descriptor, which in turn depends on the properties of the input object. The proposed method works very well without descriptors, yet it is also very flexible, so that if more information about the input object is available, the method can be easily extended to use this information in its favor. Therefore, any descriptor can be used as the additional information for the method, as will be shown, but in general, this is not necessary. This altogether makes the proposed method very robust, flexible and widely applicable. We will also show that the method is mostly superior to other existing methods in terms of robustness, accuracy and speed. Although we only consider a 3D case, the proposed symmetry measure and the method built on it can easily be extended into more dimensions (or fewer, if needed) and the symmetry measure can also be generalized for different types of symmetry.

6.1 Symmetry Measure

We again represent a general plane P as described in Section 2.4.3 and we denote $\mathbf{p} = [a, b, c, d]^T$ a 4D vector of the plane coefficients. In this chapter we do not implicitly consider the plane coefficients to be normalized in any way. A vector function $\mathbf{r}(\mathbf{p}, \mathbf{x}) \in E^3$ that reflects a point $\mathbf{x} = [x, y, z]^T \in E^3$ over a plane P represented by $\mathbf{p}$ is defined as

$$\mathbf{r}(\mathbf{p}, \mathbf{x}) = \mathbf{x} - 2\frac{\mathbf{n}_\mathbf{p}^T\mathbf{x} + d}{\mathbf{n}_\mathbf{p}^T\mathbf{n}_\mathbf{p}}\mathbf{n}_\mathbf{p}$$

where $\mathbf{n}_\mathbf{p} = [a, b, c]^T$ is the normal vector of the plane P. The components of the function $\mathbf{r}(\mathbf{p}, \mathbf{x})$ are continuous and differentiable w.r.t. $\mathbf{p}$ except for $\mathbf{p} = [0, 0, 0, d]^T$, which does not represent a valid plane. Consider a set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}, \mathbf{x}_i \in E^3, i = 1, ..., n$ which represents a sampled 3D object. In theory, X can contain samples acquired from arbitrary 3D

object representations, including volumetric ones, and all that follows is applicable regardless of the original representation from which the point set was extracted. However, we only experimented with surface data so in this work we consider X to represent a sampled 3D surface, e.g. vertices of a polygonal mesh, samples of a parametric surface, points of a raw point cloud, etc. We propose a symmetry measure that gives an evaluation of how much the point set X is symmetrical with respect to a given plane P, represented by $\mathbf{p}$. The measure is defined as

$$s_X(\mathbf{p}) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \varphi(||\mathbf{r}(\mathbf{p}, \mathbf{x}_i) - \mathbf{x}_j||). \tag{6.1}$$

The function $\varphi(l)$ is some radial function such that $\varphi(0) = 1$ and its value decreases and approaches 0 as l increases, w_{ij} are weights of point pairs and will be discussed bellow. By default, the weights are not used, i.e. $w_{ij} = 1$ for all i, j. The function φ is called a similarity function because it transforms distance of two points into their similarity. The symmetry measure s_X considers all possible pairs of points in X. For each pair $\mathbf{x}_i, \mathbf{x}_j \in X$ the point $\mathbf{x}_i$ is reflected over the plane P, represented by $\mathbf{p}$, and its distance from $\mathbf{x}_j$ is computed and transformed into similarity using φ. These similarities are summed together for all pairs, giving the symmetry measure value. The idea behind the symmetry detection method proposed below is that maximizing the symmetry measure $s_X(\mathbf{p})$ for $\mathbf{p}$ should force as many points as possible to reflect over P as close as possible to other points. This idea well fits in the nature of the task of global approximate symmetry detection, which is mainly to balance the size of the matching parts of the object (how many points reflect close to other points) and the precision of the match (how close to other points they reflect).

The idea of using some form of symmetry measure to find symmetries in shapes is not new (see e.g. [104, 77, 46]). However, the symmetry measure we propose provides several significant advantages that allow for designing a very fast, robust and flexible symmetry plane detection method. First of all, if a proper φ function is used, the measure is differentiable. This is a very useful property since it allows analytically computing its gradient and provides a natural way of quickly finding its maximum using some fast gradient-based optimization method. It can also be computed efficiently, while remaining differentiable, as described bellow. Furthermore, the measure maintains the relations between every two points, which allows using the weights w_{ij} to adjust the importance of given point pairs based on additional information, if some is available, as will be demonstrated later. Although using the weights is certainly not necessary in majority of cases, in some specific situations it can be useful and this option makes the symmetry measure very flexible. The measure is well suitable for approximate or weak symmetries and it can be computed for any set of points, not requiring a closed or manifold surface or any other specific object representation. It can also be easily extended

into higher dimensions by just adding more coordinates to the vector $\mathbf{p}$ and easily generalized for different symmetry types by replacing the $\mathbf{r}$ function with a different transformation.

6.1.1 Similarity Function

The symmetry measure $s_X(\mathbf{p})$ is differentiable w.r.t. $\mathbf{p}$ (except for $\mathbf{p} = [0,0,0,d]^T$) when $\varphi(l)$ is differentiable for $l \in \langle 0;\infty)$ and $\frac{d}{dl}\varphi(0) = 0$. This holds for the Gaussian function and also for most of the Wendland's functions [101]. Although the Gaussian function is simple and easy to implement, we used the following modified Wendland's function instead

$$\varphi(l) = \begin{cases} (1 - \frac{1}{2.6}\alpha l)^5 (8(\frac{1}{2.6}\alpha l)^2 + 5\frac{1}{2.6}\alpha l + 1) & \alpha l \leq 2.6 \\ 0 & \alpha l > 2.6 \end{cases}.$$

The value α is the shape parameter of the function. The multiplier $\frac{1}{2.6}$ is our modification which ensures that the function has a similar shape and spread to the Gaussian ($e^{-(\alpha l)^2}$) for the same value of α. The main difference between the Gaussian and our Wendland's function is that the Wendland's function equals 0 for $\alpha l > 2.6$. This means that the contribution of any point $\mathbf{x}_i \in X$ to the value of $s_X(\mathbf{p})$ is fully determined by the points of X that are not farther than $\frac{2.6}{\alpha}$ from $\mathbf{r}(\mathbf{p}, \mathbf{x}_i)$.

If no two points of X are closer than $2\frac{2.6}{\alpha}$ then the contribution is determined by at most one point of X which is the closest to $\mathbf{r}(\mathbf{p}, \mathbf{x}_i)$ and the maximum value of this contribution is 1. Therefore, the maximum possible value of $s_X(\mathbf{p})$ is n and it can only occur in the case when each point of X reflects over P precisely to another point of X, which can only happen in the case of perfect symmetry. This implies that if we set $\alpha \geq 2\frac{2.6}{l_{min}}$, where l_{min} is the smallest distance between two points of X, and X is perfectly symmetrical, then the global maximum of $s_X(\mathbf{p})$ will be always in the plane of perfect symmetry.

However, with such a value of α, the function $s_X(\mathbf{p})$ would have many local maxima, especially if l_{min} was small, making it difficult to optimize. Also, in practice, objects are never perfectly symmetrical and we therefore aim to detect approximate symmetries rather than perfect ones, so we instead set α according to the size of the input object as $\alpha = \frac{15}{l_{avrg}}$, which in most cases makes the span of φ considerably larger and $s_X(\mathbf{p})$ smoother and easier to optimize. The value l_{avrg} is the average distance of the points in X from their centroid and the value 15 was chosen as approximately optimal based on vast experiments. We also note that the proposed symmetry detection method is not very sensitive to this value, as will be shown in Section 6.4. Nevertheless, in case of finding perfect symmetry, setting $\alpha = 2\frac{2.6}{l_{min}}$ could still be used for some final refinement but we do not include this into our symmetry detection method.

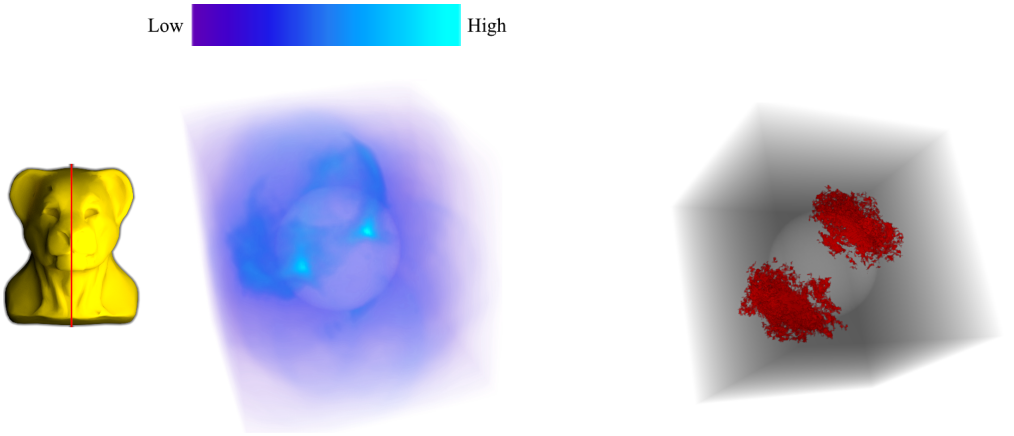
6.1.2 Efficient Computation

A brute force computation of $s_X(\mathbf{p})$ has time complexity of $\mathcal{O}(n^2)$ but for many pairs $\mathbf{x}_i, \mathbf{x}_j \in X$ the similarity $\varphi(||\mathbf{r}(\mathbf{p}, \mathbf{x}_i) - \mathbf{x}_j||)$ is 0, so it only needs to be computed for pairs where $||\mathbf{r}(\mathbf{p}, \mathbf{x}_i) - \mathbf{x}_j|| \leq \frac{2.6}{\alpha}$. We use a uniform grid with the cell size $\frac{2.6}{\alpha} \times \frac{2.6}{\alpha} \times \frac{2.6}{\alpha}$. During the computation of $s_X(\mathbf{p})$, after a point $\mathbf{x}_i$ is reflected over the given plane and ends up in a cell C, only points in C and cells adjacent to C are used for the symmetry measure computation. This way, due to the locality of the Wendland's function, $s_X(\mathbf{p})$ can be computed efficiently and remain first-order differentiable. Higher-order diffierentiability can easily be achieved using some higher-order differentiable Wendland's function, but in our application we only need the first-order differentiability and the function we use is less computationally expensive than the higher-order differentiable ones.

6.1.3 Simplification

Using the above described computation, the symmetry measure can well be used to quantify symmetry for a given plane even when the input object has quite a large number of points. But for the purpose of symmetry detection, where the symmetry measure must be evaluated repeatedly, it can still be too computationally expensive. However, in extensive experiments we observed that the symmetry measure well represents the symmetry of a point set even after the set gets simplified to a rather low number of points, given a proper simplification method is used. One way to simplify a set of points is to downsample it randomly but this approach is insufficient, since it results in a point set that does not represent the shape of the original object very well. Instead, we use the following simplification algorithm, which is quite simple and very fast. A 3D grid is created for the input point set with the cell size $\frac{l_{avrg}}{k} \times \frac{l_{avrg}}{k} \times \frac{l_{avrg}}{k}$ and each occupied cell gives one point of the simplified point set by averaging all points contained in the cell. It is desired to simplify the point set to approximately m points, so the simplification of the original point set is repeated several times with increasing value of k until the resulting point count reaches at least m, usually it gets slightly above m.

The left column of Figure 6.2 visualizes the symmetry measure for three objects simplified to approximately 1000 points using the above described algorithm. Each point $\mathbf{q} \in \langle -1; 1\rangle \times \langle -1; 1\rangle \times \langle -1; 1\rangle$ in the 3D space corresponds to a single plane with normal vector $\mathbf{n_p} = \frac{1}{||\mathbf{q}||}\mathbf{q}$ and $d = l_{avrg} \cdot (||\mathbf{q}|| - \frac{1}{2})$. The symmetry measure increases as the color goes from purple to light blue. The $\frac{1}{2}$ shift in the d coefficient causes the empty sphere in the middle, we only consider $d \geq 0$. For each object, the viewpoint was chosen to maximize the perceived information. For visualization simplicity, the objects are represented by triangle meshes but only their vertices were



(a) Lion
2213 points

Lion convergence

(b) Armadillo
172974 points

Armadillo convergence

(c) Partial Armadillo
64647 points

Partial Armadillo convergence

Figure 6.2: The symmetry measure visualized for Lion - (a), Armadillo [51] - (b) and its partial version - (c), simplified to approximately 1000 points (original point counts are in the captions), the symmetry measure increases as the color goes from purple to light blue, the objects are shown with the plane in the global maximum of the measure, the right column shows the convergence regions for the given objects.

used as the original point sets and input for the simplification. In Figures 6.2a and 6.2b it appears as if there are two significant maxima but this is caused by planes with normal $\mathbf{n_p}$ being the same as planes with normal $-\mathbf{n_p}$ for $d = 0$ and, in these two cases, the plane corresponding to the global maximum passes very near the origin, so d is close to 0. This causes the light spot of the global maximum to be split into two spots on the opposite sides of the empty sphere whose surface corresponds to $d = 0$. In Figure 6.2c, there is only one spot of the global maximum because the plane it represents does not pass so near the origin. Next to each visualization, there is the corresponding object together with the plane that represents the global maximum of the symmetry measure. In all three cases the global maximum corresponds to a very good symmetry plane of the given object, even though it is simplified to only approximately 1000 points. The original point counts are in the captions.

6.1.4 Locating Maxima

To find a local maximum of the symmetry measure, we employ a quasi-Newton optimization method L-BFGS [58], which uses the gradient of the symmetry measure, exploiting its differentiability, and it usually converges, to a sufficient precision, in only several iterations. Computation of the gradient can be accelerated in the same way as the computation of the symmetry measure itself (see Section 6.1.2). This is because the first-order derivative of $\varphi(l)$ also equals 0 for $\alpha l > 2.6$. We also tried using the Nelder-Mead optimization method [72], which does not use the gradient. It also worked but needed a much larger number of iterations making the optimization multiple times slower. This suggests the true usefulness of the analytically computed gradient. Furthermore, differentiability can be useful even in case of non-gradient methods, because it makes the symmetry measure free of sudden changes, providing more stability to the optimization.

Before starting the L-BFGS optimization, the point set should be translated together with the initial plane somewhere near the origin because in a large distance from the origin (large d) even a slight change of the plane normal vector direction can cause a significant change of the plane position, which can negatively influence the convergence of the optimization method. In order to find the global maximum of some function, generally the initial point needs to be somewhat close to it. The right column of Figure 6.2 shows a visualization of the plane space, for the same three objects using the same parameterization as in the top row, where the red spots correspond to a region of planes from which the L-BFGS optimization converged to the global maximum - the convergence region. The objects were again simplified to approximately 1000 points. For Lion and Armadillo, the convergence regions are very large. For partial Armadillo, due to its much weaker symmetry, the region is smaller but still considerably big, with the angle between normal

vectors of any two planes on the opposite sides of the region exceeding 45°. This indicates that in order to find the global maximum, the starting plane of the optimization does not need to be particularly close to it, even in the case of quite weakly symmetrical objects. This property of the symmetry measure will be exploited in the proposed symmetry detection method.

6.2 Proposed Symmetry Detection Method

The proposed global symmetry plane detection method roughly follows the common RANSAC scheme, which is very often used in symmetry detection and related areas. The key idea of RANSAC is generating a large number of candidate solutions based on the input data, which creates a good enough probability of a sufficiently good solution being among them. Each of the candidates is evaluated for its fitness, so creating unnecessarily large number of candidates can easily lead to a time consuming computation. But in our case, since the numerical optimization converges to the global maximum of the symmetry measure from quite a large distance, we do not need to find a solution that close to the best one. We only need to find one in the considerably big convergence region, suggesting that a rather sparse sampling of the candidate space will be sufficient. Determining with certainty if a candidate lies in the convergence region is impossible, but a good indication is the symmetry measure itself. It can be expected that planes near the global maximum have a higher symmetry measure than those far from it.

The overview of the proposed method follows. Several candidate symmetry planes are created and only a small number of them, with the largest symmetry measure, are selected as having the largest potential of being in the convergence region. Then the optimization is started from these few planes and the resulting plane with the largest final symmetry measure is selected. The other planes to which the optimizations converge can be used as secondary planes in case of objects with multiple significant symmetries, as will be described later. For reasons described in Section 6.1.4, before the symmetry detection itself, the input set of points is translated so that its centroid is at the origin and in the end the inverse translation is applied to the resulting symmetry plane(s).

6.2.1 Creating the Candidate Planes

Meaningful candidate planes could be created by taking each pair of points $\mathbf{x}_i, \mathbf{x}_j \in X$, $i \neq j$ and creating the plane of symmetry of these two points. But using this approach directly on X results in an overwhelming number of planes, at least when X consists of more than a few tens of points. Therefore, we first simplify X using the algorithm described in Section 6.1.3 with $m = 100$, creating a new set of points X_{cand} with approximately 100 to 110

points. The candidate plane creation is then performed on X_{cand}, creating approximately 5000 to 6000 candidate planes, which is still unnecessarily many. However, we observe that many of the candidates are very similar, which means there is no need to evaluate all of them.

Candidate Pruning

We use the following distance function to measure the distance between two planes represented by $\mathbf{p}_u$, $\mathbf{p}_v$

$$D(\mathbf{p}_u, \mathbf{p}_v) = \begin{cases} ||\widehat{\mathbf{p}}_u - \widehat{\mathbf{p}}_v|| & \mathbf{n}_{\mathbf{p}u}^T \mathbf{n}_{\mathbf{p}v} \geq 0 \\ ||\widehat{\mathbf{p}}_u + \widehat{\mathbf{p}}_v|| & \mathbf{n}_{\mathbf{p}u}^T \mathbf{n}_{\mathbf{p}v} < 0 \end{cases}$$

where $\widehat{\mathbf{p}} = \frac{1}{||\mathbf{n}_\mathbf{p}||}[a, b, c, \frac{d}{l_{avrg}}]^T$ and $\mathbf{n}_{\mathbf{p}u}$, $\mathbf{n}_{\mathbf{p}v}$ are the normal vectors of the planes. This is basically the D_{ED} distance function from Chapter 5 only modified for non-normalized planes. A theoretically more appropriate distance function can be derived when representing the candidates as reflection transformations as described in Chapter 5 (see also Chapter 4 and [79]). However, as also shown in Chapter 5, the distance function we use here shows very similar behavior in practice (given the object is centered, which we did at the beginning), is simpler, easier to implement and does not require conversion to matrix represented transformations. When creating the candidates, if a newly created candidate $\mathbf{p}_v$ is closer than $\delta = 0.1$ to the closest previously created candidate $\mathbf{p}_u$, the new one is not added to the candidate set. Instead, $\mathbf{p}_u$ is replaced with $avrg(\frac{1}{||\mathbf{n}_{\mathbf{p}u}||}\mathbf{p}_u, \frac{1}{||\mathbf{n}_{\mathbf{p}v}||}\mathbf{p}_v)$, computed as

$$avrg(\mathbf{p}_u, \mathbf{p}_v) = \begin{cases} \mathbf{p}_u + \mathbf{p}_v & \mathbf{n}_{\mathbf{p}u}^T \mathbf{n}_{\mathbf{p}v} \geq 0 \\ \mathbf{p}_u - \mathbf{p}_v & \mathbf{n}_{\mathbf{p}u}^T \mathbf{n}_{\mathbf{p}v} < 0 \end{cases}.$$

If $\mathbf{p}_u$ is already an average of ψ candidates, $\psi > 1$, then $\mathbf{p}_v$ is only added to the average which is computed as $avrg(\frac{1}{||\mathbf{n}_{\mathbf{p}u}||}\mathbf{p}_u, \frac{1}{\psi||\mathbf{n}_{\mathbf{p}v}||}\mathbf{p}_v)$. To accelerate locating the closest candidate, we use a 4D grid with cell size $\delta \times \delta \times \delta \times \delta$ where for each candidate $\mathbf{p}$, the vector $\widehat{\mathbf{p}}$ is stored. This is plausible because D is a Euclidean distance. The grid query is similar to the one described in Section 6.1.2 for the 3D grid, only differing in that when locating the closest candidate to $\mathbf{p}$, the query is executed for both $\widehat{\mathbf{p}}$ and $-\widehat{\mathbf{p}}$ because they represent the same plane. We observe that candidates created by averaging a very low number of planes can quite safely be considered outliers. Therefore, we further remove all candidates created by averaging less than $\psi_{min} = 4$ planes, which basically means that there are at most 3 pairs of points in X_{cand} that are roughly symmetric w.r.t. the given candidate plane, which is too few to generate a meaningful candidate. The remaining candidates represent the final candidate set, which usually consists of only a few hundreds of planes, so the candidate space is sampled rather sparsely but still sufficiently.

6.2.2 Selecting the Best Candidates

Once the candidates are created, we use the algorithm from Section 6.1.3 with $m = 1000$ to create another simplified version of X with approximately 1000 points, denoted X_{simp}. Then we compute the symmetry measure $s_{X_{simp}}$ for all the candidate planes. In case of objects with weaker or multiple less significant symmetries, it is not safe to only select the best candidate to initialize the optimization because its relatively large symmetry measure can be caused by its proximity to some significant local maximum rather than the global one. So we select S candidates with the largest symmetry measure and start the optimization from all of them. As the candidate pruning ensures that the candidates are not too similar, it is unlikely that the optimization converges to the same plane from all of them. In the end, we get S local maxima of the symmetry measure, among which we select the one with the largest symmetry measure and declare it the resulting plane of symmetry. We use $S = 5$.

6.2.3 Detecting Multiple Planes

Since the above described approach provides S local maxima of the symmetry measure the method can be easily adjusted to find multiple symmetry planes of the input object. It is very likely that two or more planes among the S local maxima will be very similar, therefore, to avoid detecting the same plane multiple times, we do the following. We sort the S planes representing the local maxima according to their symmetry measure in a descending order and we iterate through the sorted list from the first to the last plane. The first plane is always accepted as one of the resulting symmetry planes and any other plane is only accepted if its distance D from the closest already accepted plane is larger than 0.25. The method can find at most S significant symmetries this way. Therefore, when used to detect multiple planes, it is advisable to increase the value of S for the price of slightly larger time consumption. We use $S = 20$ instead of the original 5, which is sufficient in most applications. Because the local maxima can represent symmetry planes of varying significance, we further recommend accepting only planes with symmetry measure above some predefined threshold. The value of the threshold depends on the intended application but for a general purpose we recommend a value about 70% of the symmetry measure of the best detected plane.

6.2.4 Using the Weights

Until now we have been ignoring the weights w_{ij} in the symmetry measure s_X (see Equation 6.1) and considered all of them equal to 1. But when using the weights the method can be much more flexible. The weight w_{ij} can be

expressed as $w_{ij} = w^s_{ij} w^d_{ij}(\mathbf{p})$ where w^s_{ij} is a static weight and $w^d_{ij}(\mathbf{p})$ is a dynamic weight which depends on the plane P represented by $\mathbf{p}$. Now the symmetry measure s_X can be expressed as

$$s_X(\mathbf{p}) = \sum_{i=1}^{n} \sum_{j=1}^{n} w^s_{ij} w^d_{ij}(\mathbf{p}) \varphi(||\mathbf{r}(\mathbf{p}, \mathbf{x}_i) - \mathbf{x}_j||).$$

The dynamic weights can, for example, represent the symmetry of normal vectors or directions of principle curvatures in corresponding pairs of points with respect to a given plane. The static weights can be set to represent the importance of given pairs of points. In other words, the more it is desired for the point $\mathbf{x}_i$ to end up in or near the point $\mathbf{x}_j$ after reflecting it over the plane of symmetry the higher the static weight w^s_{ij} should be. The importance can be set manually by the user or as a similarity of some kind of feature function values in the given two points. As this feature function, for example, some type of curvature can be used, since it can be expected that in two symmetrical points there are similar curvature values.

In the rest of this section we show two modifications of the proposed method that use the weights in a beneficial manner. The first modification uses similarity of Gaussian curvature values to set the static weights and the symmetry of normal vectors for the dynamic weights. The Gaussian curvature as the feature function was chosen because it has previously shown itself useful for quantifying point similarity on 3D shapes [38] in context of symmetry detection. This modification will be described in Section 6.2.5. How this weighting can be used will also be shown in Section 6.3.4. The second modification only uses the static weights, which are set according to point-to-point distance, to force the method to detect a plane perpendicular to the largest dimension of the object. It will be described in Section 6.2.6. The results and possible purpose of both these modifications will be shown in Section 6.3.4.

6.2.5 Using Gaussian Curvature and Normal Symmetry

Here we describe the first modification which uses the Gaussian curvature values and normal vectors. The curvature values can help to identify similarity of features in the input object, which can be useful in case of objects with severe partiality where the whole object is completely asymmetrical but some features still exhibit symmetry. The normal vectors can further help to make the resulting symmetry more accurate. Let us now suppose that the input object is not represented by a general set of points but by a manifold triangle mesh on which the Gaussian curvature and a unit normal vector can be computed for each vertex. The normal vectors of all vertices are computed by summing the normal vectors of triangles adjacent to the

given vertex and normalizing the resulting vector. The values of Gaussian curvature are computed as described in [65].

Apart from the set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ that represent vertices of the mesh we now also have a set of unit normal vectors $N = \{\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_n\}$ and a set of Gaussian curvatures $G = \{g_1, g_2, ..., g_n\}$ where $\mathbf{n}_i$ is the unit normal vector in the point $\mathbf{x}_i$ and g_i is the Gaussian curvature in the point $\mathbf{x}_i$. It also needs to be defined how the normal vectors and the values of Gaussian curvature will be determined when the simplification algorithm is applied. When a new point of the simplified point set is created by averaging the points in a given cell of the simplification grid, its normal vector is determined by averaging the normal vectors in all points in the cell and normalizing the resulting vector. Its Gaussian curvature is taken from the point in the cell for which the absolute value of its Gaussian curvature is the largest.

Symmetry of Normal Vectors

The dynamic weights are set using the symmetry of normal vectors in vertices. In order to measure the reflectional symmetry of two unit normal vectors in two points, we first have to define a function $\mathbf{r_n}(\mathbf{p}, \mathbf{n})$ which reflects a unit normal vector $\mathbf{n}$ over the plane P represented by $\mathbf{p}$. This function is defined as

$$\mathbf{r_n}(\mathbf{p}, \mathbf{n}) = \mathbf{n} - 2\frac{\mathbf{n_p}^T\mathbf{n}}{\mathbf{n_p}^T\mathbf{n_p}}\mathbf{n_p}.$$

The symmetry of two normals $\mathbf{n}_i$ and $\mathbf{n}_j$ is defined as the similarity of $\mathbf{r_n}(\mathbf{p}, \mathbf{n}_i)$ and $\mathbf{n}_j$. To quantify such similarity we apply the similarity function $\varphi(l)$ on the angle between $\mathbf{r_n}(\mathbf{p}, \mathbf{n}_i)$ and $\mathbf{n}_j$. As φ our Wendland's function is used with the shape parameter α set as $\alpha = 4$. This value of α was chosen so that for angle $\frac{\pi}{16}$ (that is 11.25°) the similarity is approximately 0.5 because it is reasonable that only for low angles the similarity is significant (close to 1), otherwise the value could be quite close to 1 even for rather asymmetrical pairs of points. The dynamic weights $w_{ij}^d(\mathbf{p})$ are therefore defined as

$$w_{ij}^d(\mathbf{p}) = \varphi(arccos(\mathbf{r_n}(\mathbf{p}, \mathbf{n}_i)^T\mathbf{n}_j)), \quad with \quad \alpha = 4.$$

Similarity of Gaussian Curvature Values

The static weights are set using the Gaussian curvature values as follows

$$w_{ij}^s = \begin{cases} \frac{min(|g_i|,|g_j|)}{max(|g_i|,|g_j|)} & |g_i| \geq \frac{g_{avrg}}{h} \wedge |g_j| \geq \frac{g_{avrg}}{h} \wedge g_i g_j > 0 \\ 0 & otherwise \end{cases}.$$

The weight is non-zero only when both curvatures g_i and g_j have the same sign and their absolute values are both greater than the threshold $\frac{g_{avrg}}{h}$ where

g_{avrg} is the average of absolute values of Gaussian curvatures in all points and h is a constant which we set as $h = 100$. This ensures that Gaussian curvatures with very small absolute values are not considered because they are usually present in points without significant features.

Additional Changes

When such weighting is used, we do not use the default candidate pruning, because it does not reflect the importance of given point pairs defined by the weights. Instead, we use the weights to prune the set of candidate planes in a smarter way. When creating a candidate plane P, represented by $\mathbf{p}$, as a symmetry plane of $\mathbf{x}_i \in X_{cand}$ and $\mathbf{x}_j \in X_{cand}$, we test whether the weights w^s_{ij} and $w^d_{ij}(\mathbf{p})$, computed using N_{cand} and G_{cand}, are above certain thresholds and if not, the given plane is not considered a candidate anymore. The sets N_{cand} and G_{cand} are the sets N and G (respectively) corresponding to the simplified point set X_{cand}. The normal vector information and Gaussian curvature information are quite damaged by the simplification process and therefore the thresholds for the weights should not be set very large. Specifically we consider the plane P, created as a symmetry plane of $\mathbf{x}_i \in X_{cand}$ and $\mathbf{x}_j \in X_{cand}$, a candidate plane if $w^s_{ij} > 0$ and $w^d_{ij}(\mathbf{p}) > 0.25$. Also, since with this weighting the symmetry measure gets more complex (less smooth) and more difficult to optimize, we use $m = 200$ instead of $m = 100$, when simplifying X to create X_{cand}, to sample the space of candidate planes more densely.

6.2.6 Using Point-to-Point Distance

Here we describe the second modification that uses the point-to-point distances to set the weights. There are many objects which are symmetrical with respect to more than one plane and in some cases only some of these planes are desired to be detected. For example, imagine an object with considerably different sizes in different dimensions (width, height, depth). In such cases, the user might want to always detect the plane along the smaller dimensions and perpendicular to the largest dimension to maximize the object's span across the detected symmetry plane. In order to achieve such behavior a very simple weighting can be applied by setting the static weights to reflect the distance between the corresponding points. Specifically, we set them as $w^s_{ij} = \frac{||\mathbf{x}_i - \mathbf{x}_j||}{l_{avrg}}$. The dynamic weighs remain unused, i.e. $w^d_{ij}(\mathbf{p}) = 1$. When such weights are plugged into the symmetry measure, pairs of points with larger mutual distance will be treated as more important. Therefore, the method will try to find such a symmetry plane for which there is the best symmetry between the pairs with the largest distance. The rest of the method remains unmodified in this case. Similar weighting could also be used to achieve the opposite effect and force the method to detect planes

along the largest dimension and perpendicular to the smaller ones, but we did not perform any experiments in this way.

6.3 Results

The proposed method was implemented in *C#* and its results were acquired on a computer with CPU *Intel® Core™ i7-4770* and 16 GB of memory running a *Windows 10* operating system. Until stated otherwise, we present results of detecting the single most significant symmetry plane, because this is what our method is primarily designed for, using the basic version of the proposed method where the weights in the symmetry measure are not used ($w_{ij} = 1$). Later we will also show some results of multiple plane detection and of the two modifications mentioned above. We always use the proposed method with its default parameter values introduced throughout the previous text.

We compare our method to the View-based method [53]. Although it is not very appropriate for weakly symmetrical objects, it shows good accuracy for objects with stronger symmetry, is fast and robust to noise and works on an arbitrary triangle mesh, not requiring any further property such as manifoldness. Therefore, we consider it one of the state-of-the-art methods for symmetry plane detection. Moreover, its implementation is publicly available [52]. We always use the View-based method with its default parameter setting proposed in [53].

We further compare our method to the Clustering-based method by Shi et al. [86], which is an improved version of one of the most commonly known symmetry detection methods by Mitra et al. [67]. Apart from using a more appropriate metric in the transformation space, [86] is very similar to [67], so comparison to [86] can, in a sense, be considered a comparison to [67] as well. The Clustering-based method is designed to detect symmetries of much more general types, not just reflectional ones. We used the implementation of [86] that was adjusted by its authors themselves to specifically detect reflectional symmetries (symmetry planes). If the method found multiple symmetries, the most significant one was selected as the result according to the authors' instructions. The default parameter values were also provided to us by the authors. The Clustering-based method was run on *Linux Mint 18.3*.

There are other relevant methods for symmetry plane detection but mainly because their implementation is not available (as also mentioned by [53]), we compare to [53] and [86]. For fair comparison to the View-based and Clustering-based methods (which require a triangle mesh) and because triangle meshes are the most common way of representing 3D objects in computer graphics, the test objects we used are all triangle meshes. The basic version (with $w_{ij} = 1$) of our method only uses the vertices of each mesh

as the input set of points. However, unlike [53] and [86], the basic version of our method could as well be used for different representations including raw point clouds, since it only works with points and does not require connectivity information.

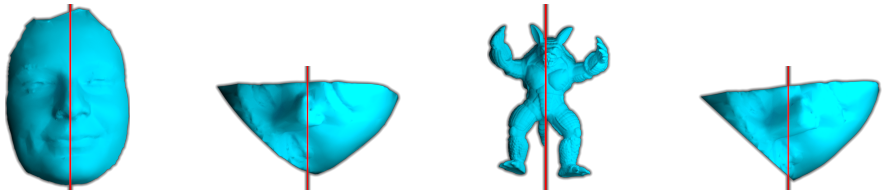
Figure 6.3 shows 26 objects of varying shapes and properties, together with their symmetry planes detected by the proposed method. The objects are always rotated so that the detected plane (marked by the line) is perpendicular to the plane of the figure. They were taken from various datasets [87, 26, 1, 51, 95] and the four faces were provided to us by the authors of the *Fidentis* project [15]. The Figures 6.3a - 6.3i contain artificial and mostly strongly symmetrical objects, 6.3j - 6.3u show realistic 3D-scanned objects and 6.3v - 6.3z objects with missing parts, usually damaged artificially by clipping, except for the Embrasure, which was damaged naturally before it was scanned. The proposed method detected correct symmetry planes for all the objects, including the incomplete ones, where detecting a naturally appearing symmetry plane is generally quite challenging.

Under each object its point count is shown and there are also the V and C letters, which stand for the View-based and Clustering-based method respectively, followed by either a check mark - ✓, in case the corresponding method provided a correct symmetry plane, mostly visually similar to the one our method detected, or a cross mark - ✗, in case the result of the method was wrong or considerably imprecise. With the Clustering-based method we further use the check mark in brackets - (✓), which means that it did not work with its default parameter configuration but we were able to find such a configuration which made it provide a plausible symmetry plane for the given object. The View-based method was only tested with its default parameter values since changing them did not lead to noticeable improvement in its results, only to larger computation time. The implementation of the View-based method crashed, for unknown reason, when used on any object with more than approximately 35000 vertices and the one of the Clustering-based method also crashed for several objects, probably due to non-manifoldness. These cases are indicated by a dash.

The View-based method detected correct symmetry planes on 11 out of 18 of the objects which its implementation was able to process. It seems to work flawlessly on the strongly symmetrical objects (9 out of 9 correct) but expectedly the objects with weaker symmetries (2 out of 5 correct) and missing parts (0 out of 4 correct) are problematic for this method. The Clustering-based method was able to correctly detect planes on 14 out of 22 of the objects its implementation processed but only on 9 of them it worked well with its default parameter configuration, on the other 5 it only worked after tuning the parameters for the specific objects. It suggests that this method is rather sensitive to its parameter setting but it shows a larger potential on weakly symmetrical and incomplete objects than the View-based method.

(a) Beetle 988 points V:✓;C:✓
(b) Cow 2903 points V:✓;C:(✓)
(c) Elephant 19753 points V:✓;C:(✓)
(d) Homer 5103 points V:✓;C:✓
(e) Mannequin 6737 points V:✓;C:(✓)
(f) Lion 2213 points V:✓;C:✓
(g) Ant 3495 points V:✓;C:✗
(h) Formula 10969 points V:✓;C:✓
(i) Starship 3099 points V:✓;C:-
(j) Face 1 2500 points V:✗;C:(✓)
(k) Face 2 2500 points V:✗;C:✓
(l) Face 3 2500 points V:✓;C:✓
(m) Face 4 2500 points V:✓;C:✓
(n) Bunny 34834 points V:✗;C:(✓)
(o) Armadillo 172974 points V:-;C:✗
(p) Buddha 543103 points V:-;C:-
(q) Lucy 750001 points V:-;C:-
(r) Column 69512 points V:-;C:✗
(s) Clipped Armadillo 135601 points V:-;C:✗
(t) Clipped Armadillo 134989 points V:-;C:✗
(u) Clipped Armadillo 64647 points V:-;C:✗
(v) Embrasure 117535 points V:-;C:-
(w) Clipped Face 1 2224 points V:✗;C:✓
(x) Clipped Face 2 1782 points V:✗;C:✗
(y) Clipped Face 3 1401 points V:✗;C:✓
(z) Clipped Face 4 1602 points V:✗;C:✗

Figure 6.3: Several objects with their symmetry planes detected by the proposed method, the symbols after the V/C letter state wheter the View-based/Clustering-based method detected a plausible plane for the given object.



(a) Face 1 View-based

(b) Clipped Face 4 View-based

(c) Armadillo Clustering-based

(d) Clipped Face 4 Clustering-based

Figure 6.4: Two examples of imprecisely - (a), (c) and two of completely incorrectly - (b), (d) detected planes of the View-based - (a), (b) and the Clustering-based - (c), (d) methods.

Figure 6.4 shows two selected fail cases of the View-based and two of the Clustering-based method - one of a noticeably imprecise detection and one of a completely wrong detection. These results were obtained using the default parameter settings. On the Ant object the Clustering-based method found no symmetry.

The results imply that the proposed method is considerably more robust than both the View-based and Clustering-based methods, mainly when used on objects with weaker symmetry and missing parts.

For objects with missing parts, comparison to [91] would be suitable because it is a state-of-the-art method for symmetry plane detection for objects with missing parts. Unfortunately, despite our effort, we were unable to acquire its implementation and there are no numerical results in [91], so no fair comparison to this method is possible. But our method performs very well on the same challenging objects on which [91] was also tested - the objects in Figures 6.3r, 6.3s, 6.3t, 6.3u and 6.3v. However, [91] requires a manifold triangle mesh and, due to the use of the Heat Kernel descriptor, does not work on featureless objects. These are quite constraining properties that make the method [91] much less general and less applicable than our method.

To measure the error of a detected symmetry plane, we apply the Metro [21] distance measure to evaluate the distance between the original object and the object created by reflecting the original one over the detected plane. The same error measurement was used by the authors of the View-based method in [53]. The Metro distance provides two values, the mean and the max error. The max error is the Hausdorff distance, which is very sensitive to outliers and often shows larger error for correct symmetry planes than for obviously incorrect ones. For this reason, we consider the max error inappropriate for measuring symmetry detection accuracy and we only use the mean error. Table 6.1 shows the error values of the symmetry planes detected by the View-based and Clustering-based methods and by the proposed method

Table 6.1: Comparison of the proposed, View-based [53] and Clustering-based [86] methods in terms of Metro [21] mean error, the lowest error for each object is marked bold, the second lowest is in italic, ∞ indicates no symmetry was found, the average error excludes Ant and Starship.

Object	Metro mean error			
	View-based method	Clustering-based method	Proposed method (aligned objects)	Proposed method (rotated objects)
Beetle	0.00619	*0.00586*	**0.00055**	**0.00055**
Cow	0.00730	(0.00116)	**0.00047**	*0.00072*
Elephant	**0.00030**	(0.00357)	*0.00035*	0.00046
Homer	0.00125	0.00627	**0.00090**	*0.00097*
Mannequin	0.00960	**(0.00838)**	*0.00905*	**0.00838**
Lion	0.00717	0.00368	*0.00107*	**0.00101**
Ant	0.00138	∞	**0.00042**	*0.00055*
Formula	0.00124	0.00523	**0.00035**	*0.00044*
Starship	0.00424	N/A	**0.00154**	*0.00157*
Face 1	0.03086	(0.01778)	*0.00815*	**0.00812**
Face 2	0.05604	0.02569	*0.01156*	**0.01191**
Face 3	0.01704	0.01922	**0.00955**	*0.00943*
Face 4	0.01716	0.02245	*0.00571*	**0.00567**
Average	0.01401	0.01084	*0.00434*	**0.00433**

on several objects from Figure 6.3. Since the proposed method is not fully rotation invariant, we present two error values for each object - one for an axis-aligned object and another for a randomly rotated version of the same object. These two values are always very similar, implying that the accuracy of the proposed method does not depend much on the orientation of the object. The errors in brackets were obtained after parameter tuning consistently with what is stated in Figure 6.3. The proposed method shows mostly the lowest error (usually several times lower than the other two methods), except for the Elephant, where the View-based method is slightly more precise, and the Mannequin, where the Clustering-based method shows slightly smaller error than the proposed method on the aligned version, but it was achieved after parameter tuning, without which it did not find a plausible plane. The average error in the bottom row does not include Ant and Starship and it is considerably smaller for the proposed method than for the other two methods.

Table 6.2 shows the running times of the proposed method and the View-based and Clustering-based (with default parameters) methods for several objects. Since some parts of our method can easily be parallelized, we

Table 6.2: Running times (in seconds) of the proposed, View-based [53] and Clustering-based [86] methods, the shortest time in each row is marked bold, the second shortest is in italic.

Object	Point count	Time [s]			
		View-based method	Clustering-based method	Proposed method single-thread	Proposed method multi-thread
Beetle	988	*0.45*	16.9	0.54	**0.25**
Lion	2213	*0.51*	18.0	*0.51*	**0.23**
Face 1	2500	0.50	18.9	*0.45*	**0.21**
Face 2	2500	0.53	20.6	*0.46*	**0.22**
Face 3	2500	0.50	19.3	*0.48*	**0.23**
Face 4	2500	*0.40*	18.1	0.51	**0.27**
Cow	2903	*0.50*	21.7	0.63	**0.31**
Starship	3099	0.53	N/A	*0.50*	**0.23**
Ant	3495	*0.55*	17.6	0.65	**0.29**
Homer	5103	0.63	19.1	*0.47*	**0.23**
Manneq.	6737	0.70	30.1	*0.42*	**0.22**
Formula	10969	0.78	23.7	*0.66*	**0.30**
Elephant	19753	1.11	32.7	*0.46*	**0.25**
Bunny	34834	2.06	145.6	*0.58*	**0.25**
Armadillo	172974	N/A	1190.0	*0.64*	**0.38**
Buddha	543103	N/A	N/A	*0.78*	**0.55**
Lucy	750001	N/A	N/A	*0.90*	**0.68**

show times of both single-thread and multi-thread implementations. The Clustering-based method is incomparably slower than the other two methods, so we further exclude it from the timing comparison. The graph in Figure 6.5 shows timing comparison of the proposed method to the View-based method with respect to the point count. The multi-threaded implementation of our method is faster than the View-based method. The single-threaded version is comparable to the View-based method for objects with point count up to approximately 10000 but after this point the running time of our method grows much less, which is due to the fast simplification (see Section 6.1.3). Unfortunately, the running times of the View-based method for the last three objects are not available, however, [53] states that the method processed an object with 467252 points in $48.2s$ on a computer with CPU *Intel® Xeon® X5675* (clock rate 3.07 GHz) which is roughly comparable to our machine. Since our method processed an object with 750001 points in $0.9s$, this suggests that for objects with larger point count our method is significantly faster than the View-based method.

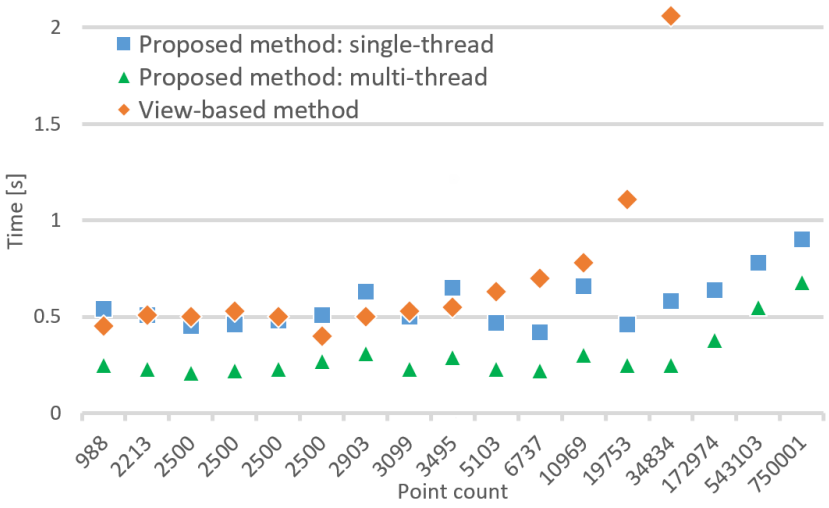


Figure 6.5: Comparison of the running times of the proposed method and the View-based method [53].

6.3.1 Noisy Objects

We created noisy versions of four different objects by adding a random vector $[rand_x, rand_y, rand_z]^T \cdot l_{avrg} \cdot mag$ to each point of the object, where $rand_x$, $rand_y$ and $rand_z$ are uniform random values from $\langle -1; 1 \rangle$ and mag is a constant which determines the noise magnitude. Figure 6.6 shows the symmetry planes detected by the proposed method on these objects, where the noise was created with $mag = 0.05$ for the objects in the top row, and with $mag = 0.1$ for those in the bottom row. Although the noise is very strong, the detected symmetry planes are still visually correct. The Clustering-based method found a plausible plane only for the Beetle with $mag = 0.05$ but only after parameter tuning, suggesting that this method is not suitable for objects with significant noise, which is also admitted in [86]. The View-based method only failed on the Lion with $mag = 0.05$, which can be considered an exception, implying that this method is comparable to our method in robustness to noise. The failure of the View-based and two selected failures of the Clustering-based method are depicted in Figure 6.7.

In the above mentioned cases the noise was uniformly distributed across the whole surface, so the random deviations of all points approximately cancel out throughout the object and its symmetry actually does not suffer that much. Figure 6.8 depicts the same four objects with added noise with $mag = 0.05$ but the noise is only added to one half of each object. The top row shows the planes detected by the proposed method, which are all visually correct, and the bottom row shows the planes detected by the View-based method, which are noticeably inaccurate. These results imply that the proposed method is more robust to non-uniformly distributed noise than the View-based method and, therefore, more robust to noise in general.

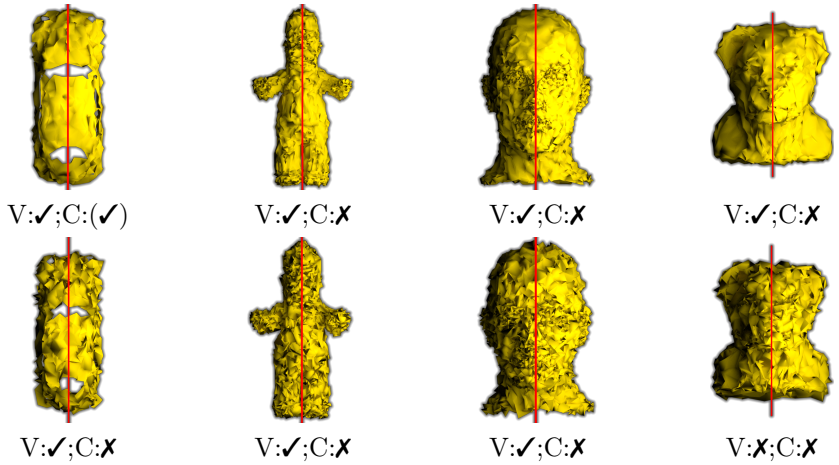
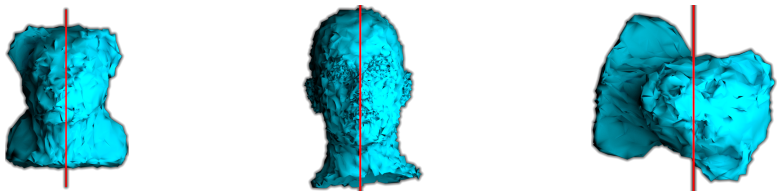


Figure 6.6: The symmetry planes detected by the proposed method on four objects with added noise with $mag = 0.05$ - top row, and with $mag = 0.1$ - bottom row.



(a) Lion
$mag = 0.05$
View-based

(b) Mannequin
$mag = 0.05$
Clust.-based

(c) Lion
$mag = 0.05$
Clust.-based

Figure 6.7: Two imprecisely - (a), (b) and one incorrectly - (c) detected planes on the noisy objects by the View-based - (a) and the Clustering-based - (b), (c) methods.

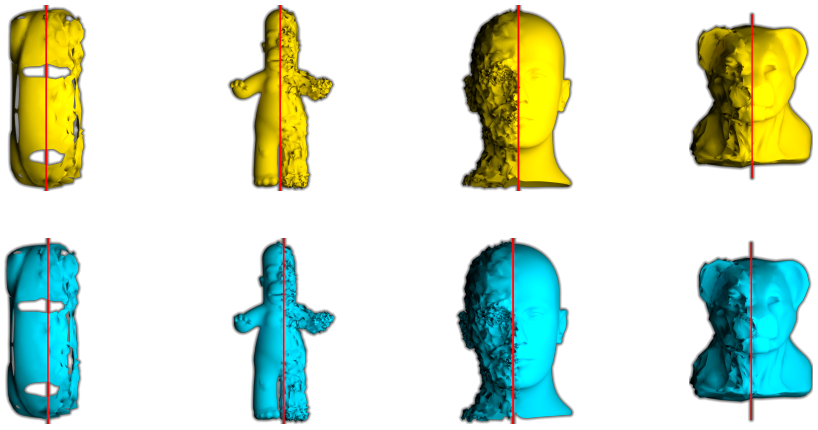


Figure 6.8: The symmetry planes detected by the proposed method - top row, and by the View-based method - bottom row, on four objects with non-uniformly distributed noise.

6.3.2 Tests on a Larger Dataset

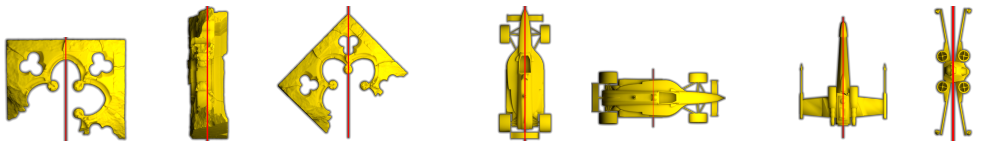
We tested our method on the *Thingi10K* [105] dataset, which consists of 10000 3D objects represented by triangle meshes. Results of our method for all objects in the dataset can be found here [41]. From the dataset we extracted all objects that could be processed by the implementations of both the View-based and Clustering-based methods (manifold with up to 30000 vertices). We randomly selected 100 objects from this set and replaced objects without any significant reflectional symmetry with new random ones, which we repeated until there were no objects to replace. We ran the proposed, View-based and Clustering-based (with default parameters) methods for all the resulting 100 objects and for each method we manually observed the results and counted correct detections. The results are in Table 6.3. Cases where the detected plane captured a significant symmetry are marked as Correct. Those that were quite close to some correct symmetry plane, but with a noticeable imprecision, are marked as Imprecise, and those that were completely wrong as Incorrect. The incorrect detections also include cases where the Clustering-based method found no symmetry (12 cases). The proposed method shows the highest number of correct detections with 3 incorrect and 2 imprecise detections. This is caused by some objects in the dataset being heavily undersampled in some areas and oversampled in others, e.g. typical CAD models that often contain densely sampled details but their planar or otherwise monolithic areas are only covered by large triangles with no or very few vertices. Since for our method we only used the vertices and not the triangles, this property sometimes forced the method to detect a symmetry of some small oversampled detail, which was globally incorrect. However, this can easily and quickly be resolved by uniformly sampling the surface instead of only taking the vertices as the input point set. We used the *MeshLab's* [20] *stratified triangle sampling* to extract a better quality point set for each of the 100 objects, after which our method achieved 99 correct detections (see the last column of Table 6.3).

Table 6.3: Number of Correct/Imprecise/Incorrect detections for the 100 random objects from *Thingi10k* [105].

	View-based	Clustering-based	Proposed	Proposed (sampled)
Incorrect	10	30	3	0
Imprecise	6	13	2	1
Correct	84	57	95	99

6.3.3 Detecting Multiple Planes

The method can also be used to detect multiple symmetries, as described in Section 6.2.3. We used it on three different objects and only accepted planes with symmetry measure larger than 70% of the best plane's symmetry measure. The objects with the detected planes are shown in Figure 6.9. The number under each figure denotes the symmetry measure of the plane relative to the measure of the best plane for the given object. The computation times in the caption are of the single-/multi-thread implementation.



(a) 1.00 (b) 0.84 (c) 0.78 (d) 1.00 (e) 0.72 (f) 1.00 (g) 0.90

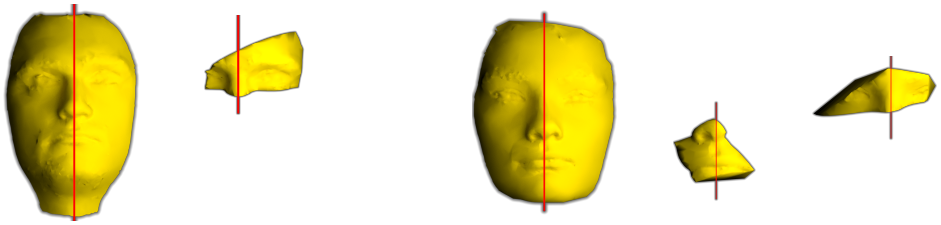
Figure 6.9: All the planes detected by the proposed method on the Embrasure - (a), (b), (c) (time: 0.89/0.40 s), Formula - (d), (e) (time: 1.66/0.58 s) and Starship - (f), (g) (time: 0.84/0.32 s).

6.3.4 Results of the Modified Versions

We mentioned two modifications of our method in Section 6.2.4 which both employ the weights in the symmetry measure in some way. Their results follow.

Gaussian Curvature and Normal Symmetry Weighting

Curvatures in general are very good for detecting local features of 3D models, especially those representing human faces or human heads [49]. In the following text, we will show that the first modified version of our method, which uses the Gaussian curvature similarity and the normal vector symmetry to set the weights w_{ij}, can be used to detect the symmetry plane of very small parts of human faces, as long as at least some local features are preserved. Figure 6.10 shows the symmetry planes detected using this modification on heavily damaged versions of two of the scanned human faces. The figure also contains the original non-damaged faces for comparison. In all three cases the symmetry planes are detected correctly, despite the fact that the objects contain very little symmetry information, because there are still some symmetrical local features preserved. Also, the normal vector symmetry weighting makes the symmetry detection more accurate. When the non-weighted version was used on these objects, it found a completely incorrect plane in all cases.

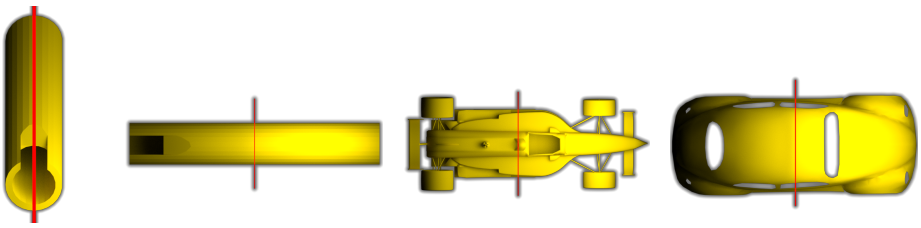


(a) (b) (c) (d) (e)

Figure 6.10: Faces 2 and 3 with symmetry planes - (a), (c) and small parts of Face 2 - (b) (980 points, time: 1.84/0.62 s) and Face 3 - (d) (1026 points, time: 1.72/0.72 s), (e) (803 points, time: 1.52/0.52 s) with symmetry planes detected by the proposed method with the Gaussian curvature and normal symmetry weighting.

Distance Weighting

The second modification, which uses point-to-point distances as weights, can be used to force the method to detect a plane along the smaller and perpendicular to the larger dimension of the input object. For example, see the cylindrical object depicted in Figure 6.11a together with the plane detected on it by the non-weighted version of the proposed method. In some cases, the user might prefer to find the plane shown in Figure 6.11b instead, although the object is slightly less symmetrical with respect to this plane. This plane is exactly the one detected by the proposed method with the distance weighting. Figures 6.11c and 6.11d depict additional two objects with the symmetry planes detected using the distance weighted modification of our method. In both cases the plane captures some non-negligible symmetry in the object (the wheels in the Formula and the fenders in the Beetle) and is also perpendicular to its largest dimension. This is despite the fact that in both cases there is one considerably stronger symmetry along the largest dimension, which the distance weighting suppresses.



(a) (b) (c) (d)

Figure 6.11: A cylindrical object and its symmetry plane detected by the non-weighted version of the proposed method - (a) and symmetry plane detected by the proposed method with the distance weighting for this and two other objects- (b), (c), (d).

6.4 Parameters

The basic version of the proposed method (with $w_{ij} = 1$, detecting a single plane) has several parameters whose default values were presented throughout this chapter at the first mention of each one. The default values were set based on extensive testing and the method does not require any parameter tuning in vast majority of cases. Nevertheless, there could be situations where the knowledge of the parameters' meaning could be useful. Most of the parameters represent the typical trade-off between reliability or accuracy and computational efficiency.

- The target point count of X_{simp} influences the precision of the detection. The larger the point count, the more accurately the symmetry measure represents the symmetry of the object and the more precise the detection is. At the same time increasing this value makes the method slower. Default is 1000.

- The target point count of X_{cand} determines the number of created symmetry candidates which grows quadratically with it. With more candidates, there is larger chance of finding one sufficiently close to the global maximum of the symmetry measure but it also takes more time to prune and evaluate them. Default is 100.

- The δ parameter determines the roughness of the candidate pruning step. With lower δ more candidates are kept, increasing chance of good candidates being among them but also making the evaluation step more timely. Lower δ also requires selecting larger value of S (see below). Default is 0.1.

- Candidates that were created by averaging less than ψ_{min} planes during the pruning step are further removed. With larger ψ_{min} less candidates are kept and the evaluation step is less timely but there is also larger chance of removing good candidates, decreasing reliability. Default is 4.

- In the end, S best candidates are selected to start the optimization from. Increasing S makes the last step more timely but also increases chance of locating the global maximum of the symmetry measure. Default is 5.

The only parameter which is not part of the efficiency-to-quality trade-off is the α spread parameter of the similarity function φ in the symmetry measure. Although its value also influences the time consumption, it is not true that changing it to make the method slower also makes it provide better results and vice versa. We set it as $\alpha = \frac{15}{l_{avrg}}$ by default and below we will show that the result of our method is not very sensitive to its value.

Above we showed that for several objects the Clustering-based method [86] requires parameter tuning. We observed that the results of the method are the most sensitive to the value of the parameter which sets the threshold for candidate pruning, let us denote it ϵ. The method creates a number of candidates by matching pairs of points and for each candidate it performs a quick relevance test and only keeps candidates that pass the test. In [86] this is done by using the candidate to transform the local neighborhood of one of the two generating points and computing its average distance to the local neighborhood of the second point. If the distance is lower than ϵ the candidate is kept, otherwise removed. In the original Clustering-based method by Mitra et al. [67] the test is done using distances in signature space represented by curvature values where the distance between the signatures of the two generating points must be lower than some threshold in order to accept the candidate. This approach, according to [86], is less robust to noise than the one they use. However, the curvature values are also computed using local neighborhoods of the points so the two approaches are essentially very similar. Nevertheless, the sensitivity to the setting of the threshold parameter implies that the result of the clustering step is very sensitive to the quality of the candidates on its input.

Figures 6.12 and 6.13 show the sensitivity of the proposed method (top row) to the value of α and of the Clustering-based method [86] (bottom row) to the value of ϵ. For both methods we used different multiples of the default parameter values. We selected two objects where the Clustering-based method works with its default parameter setting but for different objects the optimal value of ϵ can be different from the default one. For the two lowest values of ϵ the Clustering-based method does not find any symmetry for either of the two objects. Increasing ϵ makes it provide incorrect results. The proposed method detects good symmetries for all the values of α except for the lowest one where it start being imprecise because the spread of φ starts getting too large and φ ceases to well represent point similarity.

6.5 Another Application of the Symmetry Measure

In many applications (e.g. designing fields on symmetric surfaces [73]) it is desirable not only to find the symmetry plane but also to know which points are actually symmetrical with respect to the detected plane and, eventually, how much symmetry there is between them. Many of the existing methods, including the View-based and Clustering-based methods, do not implicitly provide such information. On the other hand, the symmetry measure used in the proposed method is essentially a sum of many terms where each represents the amount of symmetry between given two points, which can be used if needed. In some cases, for a given point $\mathbf{x}_i \in X$, we might want

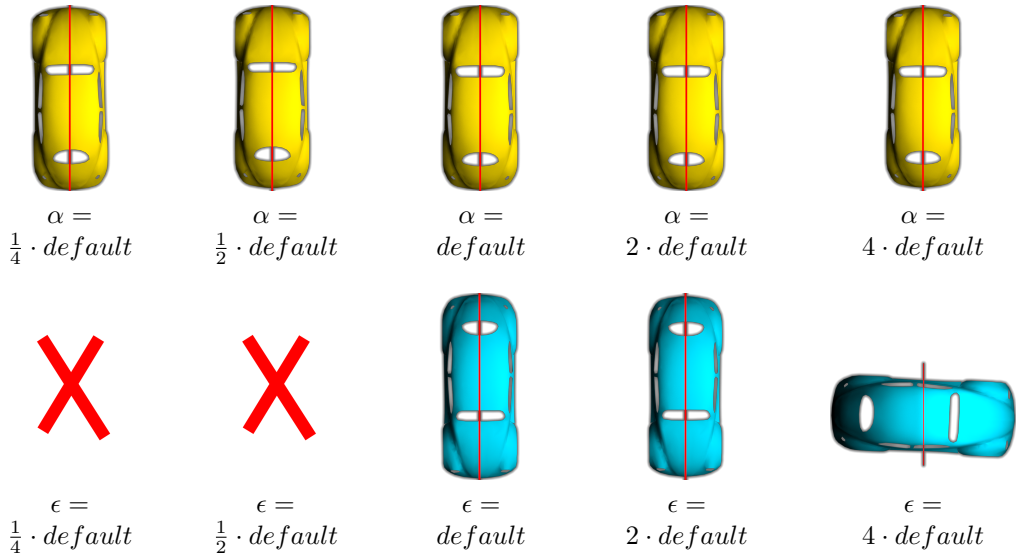


Figure 6.12: Results of the proposed method after changing the value of α - top row, and of the Clustering-based method [86] after changing the value of ϵ - bottom row, for the Beetle object.

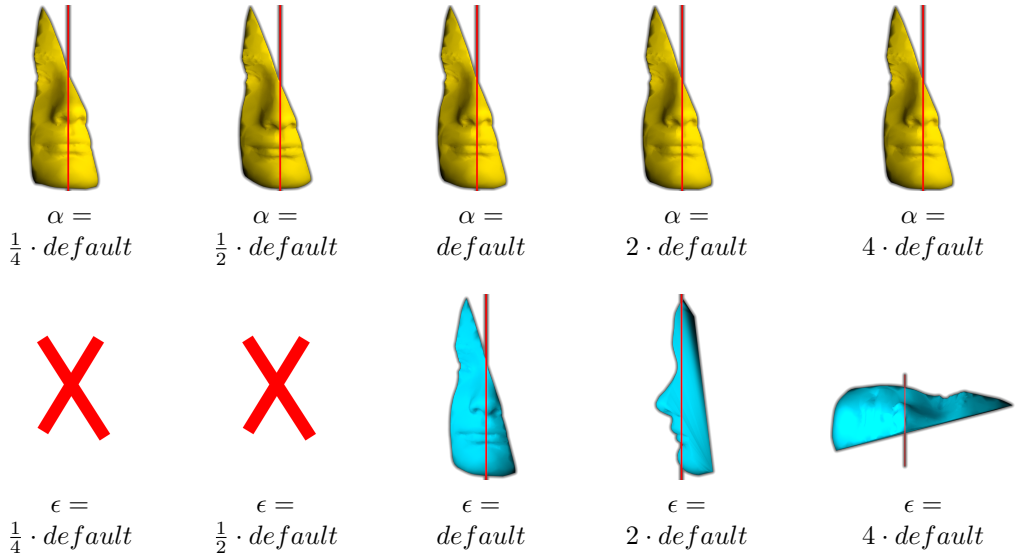
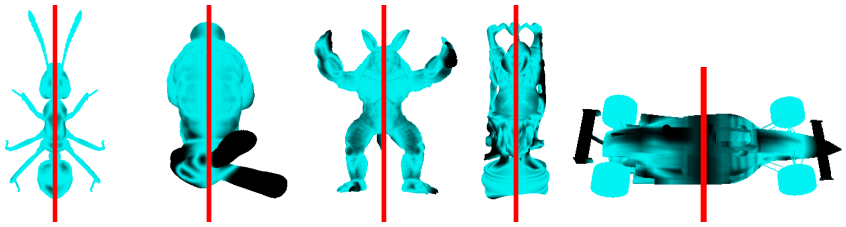


Figure 6.13: Results of the proposed method after changing the value of α - top row, and of the Clustering-based method [86] after changing the value of ϵ - bottom row, for the Clipped Face 3.

to find such a point $\mathbf{x}_j \in X$ for which there is the strongest symmetry between $\mathbf{x}_i$ and $\mathbf{x}_j$ w.r.t. a given plane P represented by $\mathbf{p}$. This only means finding such a point $\mathbf{x}_j$, for which the value $w_{ij}\varphi(||\mathbf{r}(\mathbf{p}, \mathbf{x}_i) - \mathbf{x}_j||)$ is the largest. Furthermore, if each point $\mathbf{x}_i$ is colored according to this value, the symmetry of the object can be visualized as shown in Figure 6.14. The lighter

the color, the larger the symmetry value. Figures 6.14a, 6.14b, 6.14c, 6.14d show the coloring for $w_{ij} = 1$. Figure 6.14e shows it for the case when the weights w_{ij} are set according to the distance weighting, resulting in almost no symmetry near the plane and the most symmetry in the wheels. Unlike some other methods, the concept of the proposed method also gives us a straightforward way to derive the symmetry information, with respect to any plane, for arbitrary points of the input object.



(a) (b) (c) (d) (e)

Figure 6.14: Four objects with their points colored according to the amount of symmetry with respect to their symmetry plane with no weights used - (a), (b), (c), (d) and using the distance weighting - (e) (darker - less symmetry, lighter - more symmetry).

6.6 Limitations

Although the proposed method was shown to be fast and robust in several ways, it still has some limitations and drawbacks. First, as mentioned in Section 6.3.2, the method does not handle well objects with very non-uniform point sampling. For triangle meshes this can be easily resolved using some uniform sampling technique but for raw point clouds it could be problematic and we plan to address it in the future, possibly using the weights in the symmetry measure. Second, when using the method for detecting multiple symmetries, a threshold needs to be set on the symmetry measure to only accept symmetries that are strong enough to be meaningful. There is, however, not yet a general way of determining this threshold for arbitrary input data. Finding it will be also a matter of our future efforts. Last, we showed that our method handles well objects with missing parts and objects with noise. However, our preliminary experiments show that if a single object has both significant missing parts and strong noise, the method often tends to fail which we also plan to improve later.

6.7 Summary

We proposed a new method for symmetry plane detection, based on a novel differentiable symmetry measure, and we have shown that it works very accurately for both strongly and weakly symmetrical objects and also for objects with significant noise or missing parts. We further demonstrated that in terms of robustness, accuracy and speed the proposed method outperforms the View-based [53] method and the Clustering-based [86] method which represents an improved version of the well known method of Mitra et al. [67]. Tests on a larger dataset showed that the proposed method exhibits good reliability on random objects of varying properties but they also revealed that it is not good at handling objects with significantly irregular sampling. We also showed how our method can be modified to detect multiple symmetries of a single object. The reference implementation of the proposed method is available for download at [41].

There are many possible ways of how the method can be modified, extended or generalized. The weights in the symmetry measure make the method very flexible and there are situations when using them can be very beneficial. We showed two specific ways of setting the weights and we also showed their results to prove their usefulness. In the future, we would like to further examine the possibilities of these weights and find more beneficial uses for them. We further plan to find a more efficient way of choosing a good initial plane for the local optimization. A promising possibility could be using the density peak location, described in Chapter 4 in context of surface registration, together with the knowledge about measuring distances between planes from Chapter 5. Furthermore, since the overall scheme of the method is very generic and because the symmetry measure can be used with any transformation, the method could be modified for detecting symmetries of more general types. This would also require generalizing the candidate creation process, which will be a matter of our future research. The method could also be easily modified for detecting the symmetry hyperplane in the space of arbitrary dimension, since this only requires adding more coordinates into the parameter vector of the symmetry measure.

Chapter 7

Conclusion and Future Work

In this work we focused on automatic detection of symmetries in geometric objects, mainly in the 3D space. We first provided some necessary theoretical background to understand the meaning of symmetries and described the link between symmetry detection and the problem of registration. Next, we described various existing methods and approaches for detecting symmetries of different types and briefly also for rigid surface registration. The largest part of the text then presented our own contributions to the field of rigid surface registration and symmetry plane detection.

First, we proposed a new approach to consensus evaluation for RANSAC-based registration methods using density peak location in the transformation space which was inspired by previous work in symmetry detection. We tested this approach in a model RANSAC registration algorithm and we compared it to the conventional evaluation strategy based on overlap measurement. Since the density peak location algorithm requires defining a distance metric in the space of rigid transformations, we tested and analyzed several different metrics, both practically and theoretically, and presented the results. We also proposed an improvement of the most suitable of the existing metrics and the results suggested that this new improved metric performs best.

In the context of symmetry plane detection we first focused on what we call the Mode-based approach which can actually also include the density peak evaluation algorithm we proposed previously for surface registration. We analyzed and tested various ways of representing planes in methods that implement the Mode-based approach for symmetry plane detection because the choice of the plane representation can impact the results considerably. We mainly focused on possible distance functions that can be defined for the various representations to measure similarities/dissimilarities between the planes and we reported the results. As part of this we also proposed a way of representing planes using dual quaternions and using the dual quaternion algebra to compute reflections over planes.

At last, we proposed a new method for symmetry plane detection in discrete point sets (point clouds) based on a novel differentiable symmetry

measure and a gradient-based numerical optimization. The method showed itself to be very fast, robust and also quite general. It handles well noisy objects and even objects where significant parts are missing and is mostly superior to other existing methods.

The symmetry measure used in this method contains weights of point pairs which are not used by default but we showed some of their potentially useful applications. In the future, we plan to further investigate the possibilities of these weights and use them perhaps for mitigating the influence of non-uniform distribution of points which is problematic for the method in its current state. The weights could also be used for detecting symmetries in volumetric data, rather than point clouds, by assigning static weights to pairs of voxels based on the similarity of their values.

Furthermore, although the method is very fast, the candidate evaluation step is a significant bottleneck because in this step the symmetry measure needs to be computed for each one of the several hundreds of candidate symmetry planes in order select only a few best candidates. We plan attempting to remove this bottleneck by selecting the best candidate(s) using the density peak location algorithm proposed in Chapter 4 instead, since it has already proven to work very well in the related context of registration. In order to execute this algorithm, we will need a proper representation of the space of candidate planes and a distance function in this space, for which we plan to use the knowledge from Chapter 5. This way, we will combine the material from all three major chapters in this text to design an even faster and possibly more robust method for symmetry plane detection in 3D point sets. These changes might also make the method more suitable for detection of multiple symmetries and could help to modify the method for detecting local symmetries.

Other possible future work includes generalizing the method from Chapter 6 for different types of symmetry. The symmetry measure used in the method is very generic and the reflection transformation in it can be easily replaced by any other transformation that can be parameterized differentiably. Therefore, we might be able to modify the method for detecting rigid symmetries, where the transformations could be differentiably parameterized by dual quaternions, or even general affine symmetries. This will require redesigning the process of creating candidate symmetries since neither rigid nor affine transformations are uniquely defined by a single pair of points. For this purpose, some form of a local shape descriptor could be used but a more convenient approach could be using the 4-point congruent sets [2, 64] and their affine invariant properties.

The generalizations for rigid and affine symmetries could prove rather difficult to realize. A simpler generalization could be done for detecting certain types of rotational symmetry where we could use an observation that a pair of candidate symmetry planes makes a candidate for a rotational symmetry. The idea of combining reflection planes to create candidates for

more complex symmetries might also be utilized for detecting symmetries of different types.

We would also like to study the possibility of detecting symmetries in continuous data, such as parametric curves or surfaces, by using more sophisticated approaches than simply sampling the data and using discrete methods.

Nevertheless, probably not all the future work listed above will be realized during the Ph.D. study, since some of it might prove noticeably more difficult than it seems from the current point of view. Therefore, some parts of the intended future work, such as symmetry detection on continuous data or detection of very general symmetries, could be left for future research even after the Ph.D. study.

Appendix A

Activities

A.1 Publications on International Conferences

- Hruda, L., Kolingerová, I., and Lávička, M. Plane space representation in context of mode-based symmetry plane detection. In *Computational Science – ICCS 2020* (Cham, 2020), V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, Eds., Springer International Publishing, pp. 509–523. [40] **(80%)**

A.2 Publications in Impacted Journals

- Hruda, L., Dvořák, J., and Váša, L. On evaluating consensus in ransac surface registration. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 175–186. [37] **(50%)**

- Hruda, L., Kolingerová, I., and Váša, L. Robust, fast and flexible symmetry plane detection based on differentiable symmetry measure. *The Visual Computer* (Jan 2021). [42] **(80%)**

A.3 Other Topic-Related Publications

- Hruda, L., and Dvořák, J. Estimating approximate plane of symmetry of 3d triangle meshes. In Proc. *Central European Seminar on Computer Graphics* (Smolenice, Slovakia, 2017). [38] **(50%)**

A.4 Non-Related Publications

- Hruda, L., and Kohout, J. Generic caching library and its use for vtk-based real-time simulation and visualization systems. In *VISIGRAPP*

(1: GRAPP) (2018), pp. 154–164. [39] **(90%)**

A.5 Participation in Scientific Projects

- SGS-2019-016, Synthesis and Analysis of Geometric and Computing Models, Ministry of Education, Youth and Sports

A.6 Oral Presentations

- Hledání symetrie v geometrických modelech (Language: Czech), 13. 11. 2018, seminar of the graphics group at the University of West Bohemia, Faculty of Applied Sciences, Pilsen
- Hledání symetrie v geometrických modelech (Language: Czech), 6. 12. 2018, finale of the Czechoslovakian IT SPY 2018 master thesis competition, Prague
- On evaluating consensus in RANSAC surface registration (Language: English), 2. 7. 2019, training presentation for the SGP 2019 conference and a seminar of the graphics group at the University of West Bohemia, Faculty of Applied Sciences, Pilsen
- On evaluating consensus in RANSAC surface registration (Language: English), 10. 7. 2019, Symposium on Geometry Processing 2019, Milan, Italy

A.7 Other

- 2^{nd} place in the Czechoslovakian IT SPY 2018 master thesis competition

Bibliography

[1] Aim@shape. `http://visionair.ge.imati.cnr.it/ontologies/shapes/`, 2014. Accessed: 2018-11-17.

[2] Aiger, D., Mitra, N. J., and Cohen-Or, D. 4-points congruent sets for robust surface registration. *ACM Transactions on Graphics 27*, 3 (2008), #85, 1–10.

[3] Alcázar, J. G., Hermoso, C., and Muntingh, G. Similarity detection of rational space curves. *Journal of Symbolic Computation 85* (2018), 4–24.

[4] Amato, N. M., Bayazit, O. B., Dale, L. K., Jones, C., and Vallejo, D. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)* (1998), vol. 1, IEEE, pp. 630–637.

[5] Ballard, D. H. Readings in computer vision: Issues, problems, principles, and paradigms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987, ch. Generalizing the Hough Transform to Detect Arbitrary Shapes, pp. 714–725.

[6] Belta, C., and Kumar, V. Euclidean metrics for motion generation on se(3). *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 216*, 1 (2002), 47–60.

[7] Berg, de, M., Cheong, O., Kreveld, van, M., and Overmars, M. *Computational geometry : algorithms and applications*, 3rd ed ed. Springer, Germany, 2008.

[8] Besl, P. J., and McKay, N. D. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (Feb. 1992), 239–256.

[9] Bokeloh, M., Berner, A., Wand, M., Seidel, H.-P., and Schilling, A. Symmetry detection using feature lines. *Computer Graphics Forum 28*, 2 (2009), 697–706.

[10] Bouaziz, S., Tagliasacchi, A., and Pauly, M. Sparse iterative closest point. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2013), SGP '13, Eurographics Association, pp. 113–123.

[11] Brunnström, K., and Stoddart, A. J. Genetic algorithms for free-form surface matching. In *Proc. 13th Int Pattern Recognition Conf* (1996), vol. 4, pp. 689–693.

[12] Bülow, H., and Birk, A. Spectral 6-dof registration of noisy 3d range data with partial overlap. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 35* (2013), 954–969.

[13] Caillière, D., Denis, F., Pele, D., and Baskurt, A. 3d mirror symmetry detection using hough transform. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on* (2008), IEEE, pp. 1772–1775.

[14] Censi, A., and Carpin, S. Hsm3d: Feature-less global 6dof scanmatching in the hough/radon domain. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation* (Piscataway, NJ, USA, 2009), ICRA'09, IEEE Press, pp. 1585–1592.

[15] Chalás, I., Urbanová, P., Kotulanová, Z., Jandová, M., Králík, M., Kozlíková, B., and Sochor, J. Forensic 3d facial identification software (fidentis). In *Proceedings of the 20th World Meeting of the International Association of Forensic Sciences* (2014).

[16] Charikar, M., and Guha, S. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science* (1999), FOCS '99, IEEE Computer Society, pp. 378–.

[17] Chen, Y., and Medioni, G. Object modelling by registration of multiple range images. *Image Vision Comput. 10*, 3 (Apr. 1992), 145–155.

[18] Chow, C. K., Tsui, H.-T., and Lee, T. Surface registration using a dynamic genetic algorithm. *Pattern Recognition 37*, 1 (2004), 105–117.

[19] Cicconet, M., Hildebrand, D. G., and Elliott, H. Finding mirror symmetry via registration and optimal symmetric pairwise assignment of curves: Algorithm and results. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 1759–1763.

[20] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference* (2008), vol. 2008, pp. 129–136.

[21] Cignoni, P., Rocchini, C., and Scopigno, R. Metro: measuring error on simplified surfaces. In *Computer Graphics Forum* (1998), vol. 17, Wiley Online Library, pp. 167–174.

[22] Comaniciu, D., and Meer, P. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence 24*, 5 (2002), 603–619.

[23] Combès, B., Hennessy, R., Waddington, J., Roberts, N., and Prima, S. Automatic symmetry plane estimation of bilateral objects in point clouds. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (2008), IEEE, pp. 1–8.

[24] De Castro, E., and Morandi, C. Registration of translated and rotated images using finite fourier transforms. *IEEE Trans. Pattern Anal. Mach. Intell. 9*, 5 (May 1987), 700–703.

[25] Ecins, A., Fermuller, C., and Aloimonos, Y. Detecting reflectional symmetries in 3d data through symmetrical fitting. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 1779–1783.

[26] Fang, R., Godil, A., Li, X., and Wagan, A. A new shape benchmark for 3d object retrieval. In *International Symposium on Visual Computing* (2008), Springer, pp. 381–392.

[27] Figueredo, L., Adorno, B. V., Ishihara, J. Y., and Borges, G. A. Robust kinematic control of manipulator robots using dual quaternion representation. In *2013 IEEE International Conference on Robotics and Automation* (2013), IEEE, pp. 1949–1955.

[28] Gallier, J. Notes on convex sets, polytopes, polyhedra, combinatorial topology, voronoi diagrams and delaunay triangulations. *arXiv preprint arXiv:0805.0292* (2008).

[29] Gelfand, N., and Guibas, L. J. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004), pp. 214–223.

[30] Goldman, R. *Rethinking Quaternions: Theory and Computation.* Morgan and Claypool Publishers, 2010.

[31] Hauer, M., Jüttler, B., and Schicho, J. Projective and affine symmetries and equivalences of rational and polynomial surfaces. *Journal of Computational and Applied Mathematics 349* (2019), 424–437.

[32] Hausdorff, F. *Grundzüge der Mengenlehre*. Chelsea Pub. Co, New York, 1949.

[33] Heider, P., Pierre-Pierre, A., Li, R., and Grimm, C. Local shape descriptors, a survey and evaluation. In *Proceedings of the 4th Eurographics Conference on 3D Object Retrieval* (2011), 3DOR '11, Eurographics Association, pp. 49–56.

[34] Horn, B. K. P. Extended gaussian images. *Proceedings of the IEEE 72*, 2 (1984), 1671–1686.

[35] Hruda, L. Symmetry detection in geometric models, master thesis at the university of west bohemia, faculty of applied sciences.

[36] Hruda, L., Dvořák, J., and Váša, L. On evaluating consensus in ransac surface registration: Supplementary material. `http://meshcompression.org/sgp2019`.

[37] Hruda, L., Dvořák, J., and Váša, L. On evaluating consensus in ransac surface registration. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 175–186.

[38] Hruda, L., and Dvořák, J. Estimating approximate plane of symmetry of 3d triangle meshes. In *Proc. Central European Seminar on Computer Graphics* (Smolenice, Slovakia, 2017).

[39] Hruda, L., and Kohout, J. Generic caching library and its use for vtk-based real-time simulation and visualization systems. In *VISIGRAPP (1: GRAPP)* (2018), pp. 154–164.

[40] Hruda, L., Kolingerová, I., and Lávička, M. Plane space representation in context of mode-based symmetry plane detection. In *Computational Science – ICCS 2020* (Cham, 2020), V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, Eds., Springer International Publishing, pp. 509–523.

[41] Hruda, L., Kolingerová, I., and Váša, L. Robust, fast and flexible symmetry plane detection based on differentiable symmetry measure: Supplementary material. `http://meshcompression.org/tvcj-2020`.

[42] Hruda, L., Kolingerová, I., and Váša, L. Robust, fast and flexible symmetry plane detection based on differentiable symmetry measure. *The Visual Computer* (Jan 2021).

[43] Huynh, D. Q. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision 35*, 2 (2009), 155–164.

[44] Ji, P., and Liu, X. A fast and efficient 3d reflection symmetry detector based on neural networks. *Multimedia Tools and Applications 78*, 24 (2019), 35471–35492.

[45] Kakarala, R., Kaliamoorthi, P., and Premachandran, V. Three-dimensional bilateral symmetry plane estimation in the phase domain. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 249–256.

[46] Korman, S., Litman, R., Avidan, S., and Bronstein, A. Probably approximately symmetric: Fast rigid symmetry detection with global guarantees. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 2–13.

[47] Koschat, M. A., and Swayne, D. F. A weighted procrustes criterion. *Psychometrika 56*, 2 (1991), 229–239.

[48] Kroemer, O., Amor, H. B., Ewerton, M., and Peters, J. Point cloud completion using extrusions. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)* (2012), IEEE, pp. 680–685.

[49] Kubásková, K. Recognition of important features of triangulated human head models. In *Proc. Central European Seminar on Computer Graphics* (Smolenice, Slovakia, 2016).

[50] Kuffner, J. J. Effective sampling and distance metrics for 3d rigid body path planning. In *ICRA* (2004), pp. 3993–3998.

[51] Levoy, M., Gerth, J., Curless, B., and Pull, K. The stanford 3d scanning repository. *http://www.graphics.stanford.edu/data/3Dscanrep/* (2005).

[52] Li, B. Bo li homepage. `https://sites.google.com/site/libohomepage/homepage`. Accessed: 2020-03-26.

[53] Li, B., Johan, H., Ye, Y., and Lu, Y. Efficient 3d reflection symmetry detection: A view-based approach. *Graphical Models 83* (2016), 2–14.

[54] Li, C., Wand, M., Wu, X., and Seidel, H.-P. Approximate 3D partial symmetry detection using co-occurrence analysis. In *2015 International Conference on 3D Vision* (2015), IEEE, pp. 425–433.

[55] Li, K., Yang, J., Lai, Y., and Guo, D. Robust non-rigid registration with reweighted position and transformation sparsity. *IEEE Trans. Vis. Comput. Graph. 25*, 6 (2019), 2255–2269.

[56] Li, X., Yin, Z., Wei, L., Wan, S., Yu, W., and Li, M. Symmetry and template guided completion of damaged skulls. *Computers & Graphics 35*, 4 (2011), 885–893.

[57] Lipman, Y., Chen, X., Daubechies, I., and Funkhouser, T. Symmetry factored embedding and distance. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 103.

[58] Liu, D. C., and Nocedal, J. On the limited memory bfgs method for large scale optimization. *Mathematical programming 45*, 1-3 (1989), 503–528.

[59] Makadia, A., Patterson, A. I., and Daniilidis, K. Fully automatic registration of 3d point clouds. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1* (Washington, DC, USA, 2006), CVPR '06, IEEE Computer Society, pp. 1297–1304.

[60] Martinet, A., Soler, C., Holzschuch, N., and Sillion, F. X. Accurate detection of symmetries in 3d shapes. *ACM Transactions on Graphics (TOG) 25*, 2 (2006), 439–464.

[61] Mavridis, P., Sipiran, I., Andreadis, A., and Papaioannou, G. Object completion using k-sparse optimization. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 13–21.

[62] McCrae, J., Singh, K., and Mitra, N. J. Slices: a shape-proxy based on planar sections. *ACM Trans. Graph. 30*, 6 (2011), 168.

[63] Mellado, N., Aiger, D., and Mitra, N. J. Super 4pcs fast global pointcloud registration via smart indexing. *Computer Graphics Forum 33*, 5 (2014), 205–215.

[64] Mellado, N., Aiger, D., and Mitra, N. J. Super 4pcs fast global pointcloud registration via smart indexing. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 205–215.

[65] Meyer, M., Desbrun, M., Schröder, P., and Barr, A. H. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*. Springer, 2003, pp. 35–57.

[66] Mitra, N. J., Gelfand, N., Pottmann, H., and Guibas, L. Registration of point cloud data from a geometric optimization perspective. In *Symposium on Geometry Processing* (2004), pp. 23–31.

[67] Mitra, N. J., Guibas, L. J., and Pauly, M. Partial and approximate symmetry detection for 3d geometry. In *ACM Transactions on Graphics (TOG)* (2006), vol. 25, ACM, pp. 560–568.

[68] Mudelsee, M. Estimating pearson's correlation coefficient with bootstrap confidence interval from serially dependent time series. *Mathematical Geology 35*, 6 (2003), 651–665.

[69] Nagar, R., and Raman, S. Fast and accurate intrinsic symmetry detection. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 417–434.

[70] Nagar, R., and Raman, S. Detecting approximate reflection symmetry in a point set using optimization on manifold. *IEEE Transactions on Signal Processing 67*, 6 (2019), 1582–1595.

[71] Nagar, R., and Raman, S. 3dsymm: robust and accurate 3d reflection symmetry detection. *Pattern Recognition* (2020), 107483.

[72] Nelder, J. A., and Mead, R. A simplex method for function minimization. *The computer journal 7*, 4 (1965), 308–313.

[73] Panozzo, D., Lipman, Y., Puppo, E., and Zorin, D. Fields on symmetric surfaces. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 111.

[74] Pauly, M., Mitra, N. J., Wallner, J., Pottmann, H., and Guibas, L. J. Discovering structural regularity in 3D geometry. In *ACM SIGGRAPH 2008 papers*. 2008, pp. 1–11.

[75] Petitjean, M. A DEFINITION OF SYMMETRY. *Symmetry: Culture and Science 18*, 2-3 (2007), 99 – 119.

[76] Pham, H.-L., Perdereau, V., Adorno, B. V., and Fraisse, P. Position and orientation control of robot manipulators using dual quaternion feedback. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2010), IEEE, pp. 658–663.

[77] Podolak, J., Shilane, P., Golovinskiy, A., Rusinkiewicz, S., and Funkhouser, T. A planar-reflective symmetry transform for 3d shapes. *ACM Transactions on Graphics (TOG) 25*, 3 (2006), 549–559.

[78] Pomerleau, F., Colas, F., Siegwart, R., and Magnenat, S. Comparing icp variants on real-world data sets. *Auton. Robots 34*, 3 (Apr. 2013), 133–148.

[79] Pottmann, H., Huang, Q.-X., Yang, Y.-L., and Hu, S.-M. Geometry and convergence analysis of algorithms for registration of 3d shapes. *International Journal of Computer Vision 67*, 3 (2006), 277–296.

[80] Prantl., M., Váša., L., and Kolingerová., I. Symmetry-aware registration of human faces. In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP,* (2019), INSTICC, SciTePress, pp. 185–192.

[81] Proskova, J. Discovery of dual quaternions for geodesy. *J Geom Graph 16*, 2 (2012), 195–209.

[82] Rusinkiewicz, S. Estimating curvatures and their derivatives on triangle meshes. In *Symposium on 3D Data Processing, Visualization, and Transmission* (Sept. 2004).

[83] Rusinkiewicz, S., and Levoy, M. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)* (June 2001).

[84] Rusu, R. B., Blodow, N., and Beetz, M. Fast point feature histograms (fpfh) for 3d registration. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation* (Piscataway, NJ, USA, 2009), ICRA'09, IEEE Press, pp. 1848–1853.

[85] Schiebener, D., Schmidt, A., Vahrenkamp, N., and Asfour, T. Heuristic 3d object shape completion based on symmetry and scene context. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on* (2016), IEEE, pp. 74–81.

[86] Shi, Z., Alliez, P., Desbrun, M., Bao, H., and Huang, J. Symmetry and orbit detection via lie-algebra voting. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 217–227.

[87] Shilane, P., Min, P., Kazhdan, M., and Funkhouser, T. The princeton shape benchmark. In *Shape modeling applications, 2004. Proceedings* (2004), IEEE, pp. 167–178.

[88] Simari, P., Kalogerakis, E., and Singh, K. Folding meshes: Hierarchical mesh segmentation based on planar symmetry. In *Symposium on geometry processing* (2006), vol. 256, pp. 111–119.

[89] Sipiran, I. Analysis of partial axial symmetry on 3D surfaces and its application in the restoration of cultural heritage objects. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (2017), pp. 2925–2933.

[90] Sipiran, I. Completion of cultural heritage objects with rotational symmetry. In *Proceedings of the 11th Eurographics Workshop on 3D Object Retrieval* (2018), pp. 87–93.

[91] Sipiran, I., Gregor, R., and Schreck, T. Approximate symmetry detection in partial 3d meshes. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 131–140.

[92] Smeets, D., Keustermans, J., Hermans, J., Vandermeulen, D., and Suetens, P. Feature-based piecewise rigid registration in 2-d medical images. In *ISBI* (2012), IEEE, pp. 696–699.

[93] Speciale, P., Oswald, M. R., Cohen, A., and Pollefeys, M. A symmetry prior for convex variational 3d reconstruction. In *European Conference on Computer Vision* (2016), Springer, pp. 313–328.

[94] Sun, C., and Sherrah, J. 3d symmetry detection using the extended gaussian image. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 2 (1997), 164–168.

[95] T., T., and Rieke-Zapp, D. PRESIOUS 3d cultural heritage differential scans, 2013.

[96] Tevs, A., Huang, Q., Wand, M., Seidel, H.-P., and Guibas, L. Relating shapes via geometric symmetries and regularities. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 1–12.

[97] Thrun, S., and Wegbreit, B. Shape from symmetry. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 2, IEEE, pp. 1824–1831.

[98] Vedaldi, A., and Soatto, S. Quick shift and kernel methods for mode seeking. In *European conference on computer vision* (2008), Springer, pp. 705–718.

[99] Wang, H., and Huang, H. Group representation of global intrinsic symmetries. In *Computer Graphics Forum* (2017), vol. 36, Wiley Online Library, pp. 51–61.

[100] Wang, W., Ma, J., Xu, P., and Chu, Y. Intrinsic symmetry detection on 3d models with skeleton-guided combination of extrinsic symmetries. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 617–628.

[101] Wendland, H. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in computational Mathematics 4*, 1 (1995), 389–396.

[102] Xue, F., Lu, W., Webster, C. J., and Chen, K. A derivative-free optimization-based approach for detecting architectural symmetries from 3D point clouds. *ISPRS journal of photogrammetry and remote sensing 148* (2019), 32–40.

[103] Yianilos, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 1993), SODA '93, Society for Industrial and Applied Mathematics, pp. 311–321.

[104] Zabrodsky, H., Peleg, S., and Avnir, D. Symmetry as a continuous feature. *IEEE Transactions on pattern analysis and machine intelligence 17*, 12 (1995), 1154–1166.

[105] Zhou, Q., and Jacobson, A. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).

[106] Zhou, Q., Park, J., and Koltun, V. Fast global registration. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II* (2016), pp. 766–782.