Symmetry in 3D Shapes - Analysis and Applications to Model Synthesis

Dissertation zur Erlangung des Grades des Doktors der Ingenieurwissenschaften der Naturwissenschaftlich-Technischen Fakultäten der Universität des Saarlandes

Vorgelegt durch:
Martin Bokeloh
Max-Planck-Institut Informatik
Campus E1 4
66123 Saarbrücken
Germany
am 4. August 2011

Betreuender Hochschullehrer - Supervisor
Prof. Dr. Hans-Peter Seidel, MPI Informatik, Saarbrücken, Germany

Gutachter - Reviewer
Prof. Dr. Hans-Peter Seidel, MPI Informatik, Saarbrücken, Germany
Dr. Michael Wand, Universität des Saarlandes & MPI Informatik, Saarbrücken, Germany
Prof. Dr. Vladlen Koltun, Stanford University, Stanford, United States

Dekan - Dean
Prof. Dr. Holger Hermanns, Universität des Saarlandes, Saarbrücken, Germany

Kolloquium - Examination

Datum - Date
29. November 2011

Vorsitzender - Chair
Prof. Dr. Philipp Slusallek, Universität des Saarlandes, Saarbrücken, Germany

Prüfer - Examiners
Prof. Dr. Hans-Peter Seidel, MPI Informatik, Saarbrücken, Germany
Dr. Michael Wand, Universität des Saarlandes & MPI Informatik, Saarbrücken, Germany

Protokoll - Reporter
Dr. Nils Hasler, MPI Informatik, Saarbrücken, Germany

Abstract

Symmetry is an essential property of a shapes' appearance and presents a source of information for structure-aware deformation and model synthesis. This thesis proposes feature-based methods to detect symmetry and regularity in 3D shapes and demonstrates the utilization of symmetry information for content generation. First, we will introduce two novel feature detection techniques that extract salient keypoints and feature lines for a 3D shape respectively. Further, we will propose a randomized, feature-based approach to detect symmetries and decompose the shape into recurring building blocks. Then, we will present the concept of *docking sites* that allows us to derive a set of shape operations from an exemplar and will produce similar shapes. This is a key insight of this thesis and opens up a new perspective on inverse procedural modeling. Finally, we will present an interactive, structure-aware deformation technique based entirely on regular patterns.

Kurzzusammenfassung

Symmetrie ist eine essentielle Eigenschaft für das Aussehen eines Objekts und bietet eine Informationsquelle für strukturerhaltende Deformation und Modellsynthese. Diese Arbeit beschäftigt sich mit merkmalsbasierter Symmetrieerkennung in 3D-Objekten und der Synthese von 3D-Modellen mittels Symmetrieinformationen. Zunächst stellen wir zwei neue Verfahren zur Merkmalserkennung vor, die hervorstechende Punkte bzw. Linien in 3D-Objekten erkennen. Darauf aufbauend beschreiben wir einen randomisierten, merkmalsbasierten Ansatz zur Symmetrieerkennung, der ein Objekt in sich wiederholende Bausteine zerlegt. Des Weiteren führen wir ein Konzept zur Modifikation von Objekten ein, welches Andockstellen in Geometrie berechnet und zur Generierung von ähnlichen Objekten eingesetzt werden kann. Dieses Konzept eröffnet völlig neue Möglichkeiten für die Ermittlung von prozeduralen Regeln aus Beispielen. Zum Schluss präsentieren wir eine interaktive Technik zur strukturerhaltenden Deformation, welche komplett auf regulären Strukturen basiert.

Summary

Symmetry is an essential property of a shapes' appearance and presents a source of information for structure-aware deformation and model synthesis. This thesis proposes feature-based methods to detect symmetry and regularity in 3D shapes and demonstrates the utilization of symmetry information for content generation. First, we will introduce two novel feature detection techniques that extract salient keypoints and feature lines for a 3D shape respectively. Further, we will propose a randomized, feature-based approach to detect symmetries and decompose the shape into recurring building blocks. Then, we will present the concept of *docking sites* that allows us to derive a set of shape operations from an exemplar and will produce similar shapes. This is a key insight of this thesis and opens up a new perspective on inverse procedural modeling. Finally, we will present an interactive, structure-aware deformation technique based entirely on regular patterns.

Slippage Features: Keypoint detection aims at finding *reliable* feature points invariant to global transformations and noise. In Chapter 3, we introduce a generalized concept of geometric features that detects locally, uniquely identifiable keypoints as centroids of an area that is locally maximal constrained with respect to the auto-alignment problem e.g. registration of geometry with itself. We extend the concept to multiple scales and extract features using multi-scale mean shift clustering. By construction, the local neighborhood of a keypoint is well constrained for alignment via iterative closest points (ICP). Consequently, we use ICP residuals as similarity measure which yields robust feature comparison. We obtain reliable keypoints in larger quantities than state-of-the-art methods and demonstrate the applicability of our method in the context of global alignment of rigid or deformable objects, and symmetry detection.

Symmetry Detection Using Feature Lines: We describe a new method for detecting structural redundancy in geometric data sets in Chapter 4. Our algorithm computes rigid symmetries, i.e., subsets of a surface model that recur several times within the model differing only by translation, rotation or mirroring. We base our method on matching locally coherent constellations of feature lines on the object surfaces. This feature representation reduces the required amount of data by approximately two orders of magnitude but still carries enough information necessary to find even small symmetries. In comparison to previous work, our method is able

to detect a large number of symmetric parts without restrictions to regular patterns or nested hierarchies, which has not been achieved before for huge datasets. We apply the algorithm to a number of real world 3D scanner data sets, demonstrating high recognition rates for general patterns of symmetry.

Inverse Procedural Modeling: In Chapter 5, we present an inverse procedural modeling approach that creates shapes similar to an input exemplar. We consider local similarity, i.e., each local neighborhood of the newly created object must match some local neighborhood of the exemplar. We show that we can find explicit shape modification rules that guarantee strict local similarity by looking at the structure of the partial symmetries of the object. We systematically collect such editing operations and analyze their dependency to build a shape grammar. All of this information is derived directly from the model, without user interaction. In comparison to previous work that is limited to synthesizing shapes, we present the first approach that is able to automatically extract rules than span a class of 3D objects. Overall, we are the first to provide a general theoretical and practical framework for inverse procedural modeling of 3D objects.

Pattern-Aware Shape Deformation: We introduce a new structure-aware shape deformation technique in Chapter 6. The key idea is to detect continuous and discrete regular patterns and ensure that these patterns are preserved during free-form deformation. We propose a variational deformation model that preserves these structures, and a discrete algorithm for structure adaptation that adaptively inserts or removes repeated elements in regular patterns to minimize distortion. As a tool for such structural adaptation, we introduce sliding dockers, which represent repeatable elements that fit together seamlessly for arbitrary repetition counts. Additionally, we provide an efficient numerical framework that allows deformation with structural adaption at interactive rates. Our approach is the first interactive structure-aware free-form deformation technique that can alter the shape by changing regular patterns during deformation in order to relax stretch. Further, we introduce a formal model for structure-aware regularization that is entirely based on 1-parameter partial symmetry groups.

Zusammenfassung

Symmetrie ist eine essentielle Eigenschaft für das Aussehen eines Objekts und bietet eine Informationsquelle für strukturerhaltende Deformation und Modellsynthese. Diese Arbeit beschäftigt sich mit merkmalsbasierter Symmetrieerkennung in 3D-Objekten und der Synthese von 3D-Modellen mittels Symmetrieinformationen. Zunächst stellen wir zwei neue Verfahren zur Merkmalserkennung vor, die hervorstechende Punkte bzw. Linien in 3D-Objekten erkennen. Darauf aufbauend beschreiben wir einen randomisierten, merkmalsbasierten Ansatz zur Symmetrieerkennung, der ein Objekt in sich wiederholende Bausteine zerlegt. Des Weiteren führen wir ein Konzept zur Modifikation von Objekten ein, welches Andockstellen in Geometrie berechnet und zur Generierung von ähnlichen Objekten eingesetzt werden kann. Dieses Konzept eröffnet völlig neue Möglichkeiten für die Ermittlung von prozeduralen Regeln aus Beispielen. Zum Schluss präsentieren wir eine interaktive Technik zur strukturerhaltenden Deformation, welche komplett auf regulären Strukturen basiert.

Slippage Features: Merkmalserkennung zielt darauf ab, stabile Merkmale zu berechnen, welche invariant gegenüber globalen Transformationen und Rauschen sind. In Kapitel 3 stellen wir eine Methode zur Merkmalserkennung vor, die auf Selbstregistrierung lokaler Geometrie beruht. Wir definieren ein Merkmal als Schwerpunkt einer Region, deren Registrierung mit sich selbst wohlbestimmt ist und im lokalen Vergleich besser bestimmt ist als die Registrierungen der Nachbarregionen. Wir erweitern das Konzept auf mehrere Skalen und verwenden Mean-Shift-Clustering zur Merkmalsextraktion. Aufgrund der Definition eignen sich die Merkmalspunkte zur lokalen Registrierung auf andere Merkmale. Deshalb verwenden wir die Residuen der Registrierung als Distanzmaß und erhalten somit robuste Merkmalsvergleiche. Wir erhalten stabilere Merkmalskorrespondenzen im Vergleich mit verwandten Arbeiten und zeigen den Nutzen unserer Methode am Beispiel der globalen Registrierung von Starrkörpern oder deformierbaren Objekten sowie zur Symmetrieerkennung.

Symmetry Detection Using Feature Lines: Wir stellen einen neuen Ansatz zur Symmetrieerkennung in Kapitel 4 vor. Unser Algorithmus berechnet Symmetrien, die durch eine Starrkörpertransformation beschrieben werden können, und zerlegt dadurch ein Objekt in sich wiederholende Bausteine. Unsere Methode

basiert auf Registrierung von örtlich kohärenten Ansammlungen von Merkmalslinien. Diese Merkmalsrepräsentation reduziert den Aufwand dramatisch, beinhaltet jedoch trotzdem genug Information, um sogar kleine Symmetrien zu finden. Im Vergleich zu anderen Methoden können wir viele partielle Symmetrien finden, ohne uns auf hierarchische oder reguläre Symmetrien zu beschränken. Auf großen Datensätzen war dies mit bisher existierenden Methoden nicht möglich. Wir erreichen hohe Erkennungsraten auf gemessenen 3D-Punktwolken.

Inverse Procedural Modeling: In Kapitel 5 präsentieren wir einen Ansatz zur Ermittlung von prozeduralen Regeln aus Eingabeobjekten. Unser Ziel ist die Synthese von 3D-Objekten, die lokale Ähnlichkeit zu einem Eingabeexemplar besitzen; jede lokale Umgebung des erzeugten Objekts muss im Eingabeexemplar vorhanden sein. Wir zeigen, dass es Modifikationsoperationen gibt, die lokale Ähnlichkeit garantieren und durch Betrachtung der partiellen Symmetrien auffindbar sind. Unsere Methode extrahiert systematisch Modifikationsoperationen, analysiert deren Abhängigkeiten und erzeugt eine kontextfreie Grammatik, die ähnliche Objekte erzeugen kann. All dies kann direkt aus der Symmetriestruktur berechnet werden und benötigt keine Nutzerinteraktion. Unsere Methode erlaubt nicht nur die Synthese von ähnlichen Modellen wie in vorangegangenen Arbeiten, sondern ermöglicht außerdem direktes Ableiten von prozeduralen Regeln, die eine Klasse von ähnlichen 3D-Objekten erzeugen können. Wir bieten ein generelles theoretisches und praktisches Gerüst zur Ermittlung von prozeduralen Regeln zur Erzeugung von ähnlichen 3D-Objekten.

Pattern-Aware Shape Deformation: In Kapitel 6 stellen wir eine neue Technik zur strukturerhaltenden Deformation von 3D-Oberflächen vor. Die zentrale Idee ist, kontinuierliche und diskrete Muster zu erkennen und sicherzustellen, dass diese Muster während der Deformation erhalten bleiben. Wir schlagen ein Variationsmodell für Deformation vor, welches diese Muster erhält. Des Weiteren präsentieren wir eine Methode zur Minimierung von Verzerrungen, die adaptiv Elemente aus diskreten Mustern entfernt oder einfügt, um somit den Verzerrungen entgegenzuwirken. Hierfür zeigen wir außerdem, wie geeignete Muster gefunden und verändert werden können, die sich übergangslos verändern lassen. Zusätzlich stellen wir ein effizientes, numerisches Grundgerüst vor, das gleichzeitige Deformation und Strukturanpassung bei interaktiver Nutzung ermöglicht. Unser Ansatz ist der erste interaktive Ansatz zur strukturerhaltenden Deformation, der in der Lage ist, diskrete reguläre Muster strukturell zu verändern, um Verzerrungen zu minimieren. Des Weiteren ist dies das erste strukturerhaltende Deformationsmodell, das gänzlich auf diskreten 1-Parameter Symmetriegruppen aufgebaut ist.

Contents

List of Figures

List of Tables

1 Introduction

Everything starts somewhere, although many physicists disagree.

Terry Pratchett

Progress in computer science and a strong interest in digital representations of reality have led to a plenitude of technologies for acquisition, modeling, and visualization of digital content. Today's rendering techniques range from real-time visualization of 3D worlds on mobile phones to high-quality renderings for feature films that are almost indistinguishable from reality. However, all these techniques require digital content such as 3D models, textures, material properties, and motion data. For each individual technique, the content needs to be adapted for the particular purpose. A building, for example, is represented completely differently in navigation systems than in a feature film. With an ever increasing ability to render more complex scenes and specialized content requirement, modern computer graphics faces an uphill task how to create all this content.

There are two major ways to create digital content: Firstly, acquisition of real-world objects and secondly, manual modeling by an artist. Both ways have their own advantages and disadvantages. Acquisition is naturally restricted to existing objects. Some objects might be hard or impossible to acquire, for example precise acquisition of trees is still a challenging problem. Manual modeling requires technical and artistic skill. However, both methods usually consume a lot of time and resources. Consequently, there is a desire to reuse and modify existing digital objects in order to satisfy the increasing demand. In particular in the last decade countless 3D objects, textures, and materials have been created by talented artists. Can we reuse and recombine digital objects from existing databases in such a way that it is significantly faster than traditional modeling but of comparable aesthetical quality?

A direction to approach this is to gather and interpret structural information of existing data and try to "learn" what an object is. Of course, understanding an object as we humans do is far out of the scope of this thesis and will most likely

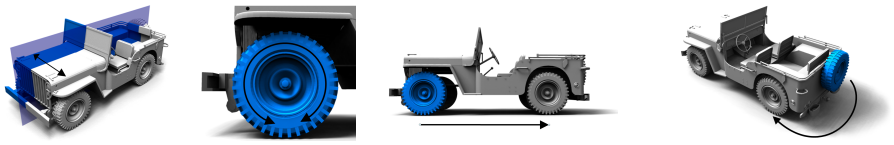
a) Maple leaf b) Snowflake c) Vitruvian Man d) Town hall Hannover

Figure 1.1: *In everyday life, we observe many different forms of symmetry. Nature itself tends to favor symmetric structures such as in a Maple leaf*[1] *or, at a much smaller scale, in the crystal structure of a snowflake*[2]*. Leonardo da Vinci's drawing* The Vitruvian Man *illustrates symmetrical properties of an idealized human male and symbolizes a strong connection between symmetry and esthetic. Especially in classical architecture symmetry played a dominant role for several centuries; the new town hall of Hannover*[3] *shows a relatively modern example from the early 20th century.*

be a research topic for future generations. But we can try to get some insight about objects from low level cues and utilize these for content creation. A fundamental source of information comes from the ability to establish correspondences between similar objects or parts of objects. This allows us to build equivalence classes of objects, helps to identify semantic components, and can give us valuable information about complex objects and their structure.

In this thesis, we focus entirely on 3D shapes and their geometric properties. This means, we have to identify correspondences between shapes or part of shapes. In particular, we are interested in correspondences *within* objects. If we are able to find a part twice within an object, we gain important information such as segmentation cues (the part may be a meaningful segment of the object) and structure information. Inspired by mathematics, the term *symmetry* has been shaped and is widely used in computer graphics to describe correspondences within an object. Symmetry has slightly different meanings in mathematics, psychology, chemistry, biology, physics, and others. In everyday speech, symmetry often refers to reflectional symmetry, for example, of a human face. However, the general term includes many other forms of symmetry that we can observe in everyday life as illustrated in Figure 1.1: The veins of the Maple leaf bear similar structures, the snowflake shows a rotational symmetry, the famous drawing *Vitruvian Man* by Leonardo da Vinci illustrates reflectional symmetry of humans, and the *New Town Hall* of Hannover contains many different forms of translational and reflectional symmetries. We will give a formal definition of symmetry in the next section.

[1] http://de.wikipedia.org/wiki/Ahorne (July 28. 2011)
[2] http://www.flickr.com/photos/39998519@N00/3211427354/ (July 28. 2011)
[3] http://de.wikipedia.org/wiki/Neues_Rathaus_(Hannover) (July 28. 2011)



a) Reflection b) Rotation c) Translation d) Rigid transformation

Figure 1.2: *Probably the most intuitive class of symmetries is rigid symmetry. The jeep is mostly reflectionally symmetric at the reflection plane in a). The wheel is rotationally symmetric b), translational symmetric to the rear wheel c), and is also symmetric to the extra wheel in the back under a rigid transformation d).*

1.1 Definition of Symmetry

Let $\mathcal{S} \in \mathbb{R}^3$ be a 3D surface, $\mathcal{P}$ a part of the surface $\mathcal{P} \subseteq \mathcal{S}$, T be a general transformation $\mathbb{R}^3 \rightarrow \mathbb{R}^3$, and $dist(S_1, S_2)$ be a distance function that measures a distance of surface S_1 to S_2.

We call $\mathcal{P}$ *symmetric* under transformation T if $dist(T(\mathcal{P}), \mathcal{S}) = 0$.

In other words, the transformed part $T(\mathcal{P})$ maps onto another part of the surface $\mathcal{S}$. Due to finite numerical precision of computers we slightly change the definition and bound the distance between $\mathcal{P})$ and $\mathcal{S}$ by $dist(T(\mathcal{P}), \mathcal{S}) \leq \epsilon$ with $\epsilon > 0$ that compensates for numerical noise (see also *approximate symmetries* in Section 1.1.2). This definition captures a large variety of symmetries and we will use specific instances for different applications. Intuitively, we consider symmetry as a *simplification* that describes a shape more compactly and simpler in some sense, and in this thesis we will stick to this thought and opt for simplicity. As a consequence, we have to restrict the class of allowed transformations because a highly complex transformation would contradict our assumption that symmetry gives simpler explanations of a shape. For example, it might be possible to deform a Maple leaf (Figure 1.1 a) into the shape of the new town hall (Figure 1.1 d). However, the conclusion "the new town hall is a Maple leaf" does not make much sense. In the following, we will discuss several classes of transformations and provide an overview of different types of symmetries that have been used in related work.

1.1.1 Transformations

- *Reflections* - A well known form of symmetry is reflectional symmetry or mirror symmetry in Euclidean space. Reflections can be parameterized by a reflection plane $(nx = d)$ where n is a normal vector on the unit sphere. Expressing the plane in spherical coordinates, we obtain 3 parameters θ, ϕ, d with $n = (sin(\phi)sin(\theta), cos(\phi)sin(\theta), cos(\theta))^\top$.

- *Rotations* - Rotational symmetries appear frequently in man-made objects

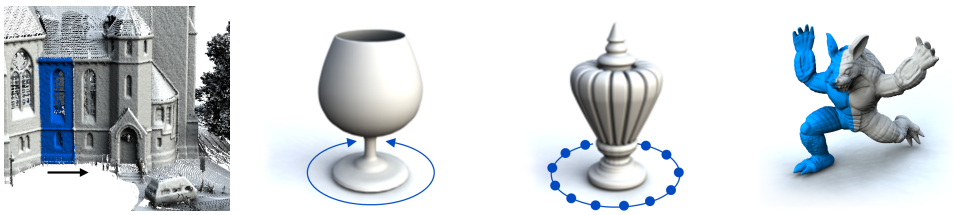
(e.g., in decorated aluminum wheels). They can be characterized by a rotation axis and a rotation angle. In total, we need 5 parameters $\theta, \phi, t_x, t_y, d_\alpha$ to specify a rotation (two angles θ and ϕ to define the rotation axis direction in spherical coordinates, two translational parameters t_x and t_y to move the rotation axis orthogonal to the axis direction and a parameter d_α to specify the rotation angle).

- *Translation* - Translational symmetries occur e.g. in buildings when a window is translationally symmetric to a neighboring window. Translations in Euclidean space are defined by a translation vector $(t_x, t_y, t_z)^\top$.

- *Rigid transformations* - A prominent example of symmetry, often referred to as *rigid symmetry*, is based on rigid transformation that include rotations, translations, and reflections. This particular type of symmetry is important for many applications because it is a very good model of manufactured objects. Industrial products usually contain building blocks of the same type that are identical up to a certain tolerance. For example, a car engine has a number of geometrically identical cylinders and pistons. Rigid symmetries model this by changing only the position and orientation of a part, which can also be done in the real world. Here, reflection is a special case that is sometimes excluded from the definition since it is physically impossible for objects like pistons to be mirrored; however, sometimes mirrored objects are manufactured. Rigid transformations without reflections are called proper rigid transformations and they are parameterized by 6 scalar values $\alpha, \beta, \gamma, t_x, t_y, t_z$ with t_x, t_y, t_z as translational part and α, β, γ as rotational part. Several notations exist on how the rotation should be interpreted. Sometimes the rotation parameters are interpreted as Euler angles: $R = R_x(\alpha)R_y(\beta)R_z(\gamma)$ where R_x represents a rotation around the x-axis (R_y, R_z respectively for y, z axis). Here, the order in which the rotations are applied is crucial and varies in the literature. Another way of interpretation is the Rodrigues'notation where $(\alpha, \beta, \gamma)^\top$ represents the unnormalized rotation axis and the norm $\left\|(\alpha, \beta, \gamma)^\top\right\|_2$ corresponds to the rotation angle [Rodrigues, 1816]. Reflections require a binary parameter. An example of different rigid symmetries is shown in Figure 1.2.

- *Similarity transformations* - Two objects are considered similar, in mathematical sense, when there exists a *similarity transformation* that transforms an object into another. A similarity transformation consists of a rigid transformation, including reflections, and a uniform scaling parameter. This type of symmetry is useful for some manufactured objects with building blocks of different scales.

- *Affine transformations* - Affine mappings are linear transformations and an additional translation: $T(x) \to Ax + t$ with $A \in \mathbb{R}^3 \times \mathbb{R}^3$ and $t \in \mathbb{R}^3$. Very few attempts have been made to detect affine symmetries.



a) Partial b) Continuous c) Regular patterns d) Intrinsic

Figure 1.3: *Variations of symmetry: a) Only a part of the building is symmetric under the shown translation whereas b), c), and d) are globally symmetric. b) Continuous symmetries have a continuous set of symmetry transformations. c) Symmetry transformations form a group in regular symmetries. d) Intrinsic symmetries consider only the intrinsic structure of a shape and not its embedding in 3D space.*

- *Others* - One can use more complex types of transformations to express symmetries. There exist various techniques that deform a shape with many degrees of freedom. Here, especially the physically motivated techniques might be of great interest since some types of symmetries are created by physical deformation, for example, a crate full of tightly packed rubber ducks. Simulation of this behavior has been investigated in detail, however, due to the usually large number of parameters it is not yet clear how to solve the inverse problem in the context of symmetry detection where neither the physical parameters are known nor any predefined objects.

1.1.2 Symmetry Types

Besides different classes of transformations, there are various types of symmetry. Here, we give a brief overview of the notations used in the computer graphics literature.

- *Global* - If a complete shape is symmetric under a transformation, we call it globally symmetric or a global symmetry of the shape. According to our definition of symmetry this means $\mathcal{P} = \mathcal{S}$. In contrast to global symmetries we call a shape partially symmetric if $\mathcal{P} \neq \mathcal{S}$. Global symmetries are easier to detect than partial symmetries because of their global influence and clearly defined domain. Partial symmetries in addition require determining the part that is symmetric.

- *Approximate* - Symmetries are called approximate or imperfect if the distance between two symmetric surfaces becomes relatively large due to different small scale features in the surface ($\epsilon \gg 0$). For example, consider Figure 1.1 d: Intuitively, we would describe the building as globally symmetric with respect to a vertical reflection plane. However, a closer examination

reveals several asymmetric parts (missing chimney on the right side, asymmetric configuration of the hands in the central clock, etc.). On a smaller scale, we would see differences in the stone surface like small cracks, vegetation, and erosion. Nevertheless, we would still think of this building as globally symmetric. Symmetries can be approximate in terms of numerical noise, acquisition noise, or even in the sense of geometric noise where the fine-scale geometry differs, as in [Mitra et al., 2006b].

- *Continuous* - A continuous symmetry refers to a symmetry with a continuous set of transformations under which a part $\mathcal{P}$ is symmetric. For example, the wine glass in Figure 1.3 b) is continuously symmetric with respect to a set of transformations $\{T \in Rot_z(p) | p \in [0, 2\pi)\}$ where $Rot_z(p)$ defines a rotation matrix with rotation angle p and rotation axis $(0, 0, 1)^\top$. For rigid transformations, this type of symmetry has been investigated by Gelfand and Guibas [2004]. In this thesis we will use the so called *slippage analysis* that computes continuous symmetries in several contexts. A more detailed introduction to slippage analysis will follow in Chapter 2.

- *Regular patterns* - Different symmetries might be linked by a regular structure in the transformations. For example, if a part $\mathcal{P}$ is symmetric under two transformations T_1 and T_2 with $T_2 = T_1^2$, we call this a regular pattern. These types of regularities appear quite often in 3D shapes and, in comparison with normal partial symmetries, they are easier to detect due to their inherent structure. We will define regular patterns more precisely in Chapter 6.

- *Intrinsic* - All transformations described so far interpret shapes from an *extrinsic* point of view; a shape lives in Euclidean space and all computations are done in 3D domain. However, shapes can also be interpreted as 2d manifolds embedded in Euclidean space. An intrinsic symmetry refers to a part $\mathcal{P}$ of a 2d manifold that can be mapped onto another part of the surface by an *isometric* deformation that preserves intrinsic distances. The armadillo in Figure 1.3 d) contains an (approximately) intrinsic symmetry, in this case a global reflection.

1.2 Rigid Symmetry as Base Model

Symmetries in general are hard to detect. Especially for symmetries that have undergone a complex deformation the process of finding this deformation *and* the instance area simultaneously leads to huge computational costs. Rigid symmetries are much more restricted but represent an excellent compromise between complexity and applicability. The reason for this is twofold: First, a rigid transformation is defined by only 6 parameters which spans a manageable search space. Second, many man-made objects are build of parts that have exact specifications and these

parts are often used multiple times (physical copies). This means that rigid symmetry is an accurate, physically-based model because it resembles the process of how these objects were constructed: Nearly identical parts are assembled into a new object by moving and rotating these parts in 3D space. Some effects are not captured in this definition like varying microscopic geometric details in parts or material deformations caused by the assembly process. Nevertheless, the model is accurate enough for our purpose.

Rigid symmetries are also a good way to explain a shape in the sense that we can derive which parts the shape was constructed from. From a programmer's perspective, a clear, well-defined structure of the problem is essential to formulate an algorithm that solves it. In our setting, an algorithm has to provide potential explanations (symmetries) and requires a formal validation criterion to accept or reject explanations. But what is a *good* explanation? Intuitively, we prefer *simple* explanations because they are easier to find and it becomes harder to create a false hypothesis with sufficient support in the data. The basic principle of simplicity is often referred to as Occam's razor and has been applied in many different fields of science. Occam's razor states: "Of two equivalent theories or explanations, all other things being equal, the simpler one is to be preferred." Transfered to symmetries this means that if we can express the same symmetry using a simpler transformation we choose the simpler one. In our context, we can also see simplicity as compression: A good explanation is more *compact* than the original shape. This does not necessarily restrict us to rigid symmetries, but demands a certain benefit for complex transformations.

Considering the compression aspect, we can also rate explanations and compare different explanations with respect to their gain in compression. This interpretation immediately leads to a clear, well-defined validation criterion that can be applied in computer programs. Please note that the goal of compression does not directly imply to be useful for content creation or shape understanding. However, it is a valuable indicator in practice and we will consider simplicity under this aspect.

In symmetry detection, and especially in the area of symmetry-aware decomposition of shapes into building blocks, we have to deal with a range of ambiguities. These can manifest themselves as variability of transformations where *different* transformations map the same part onto the same target part. In other words, the mapping between these two parts is ambiguous since we can find multiple sets of correspondences between them. We want to avoid such situations because it indicates that we did not really understand the symmetry structure of the shape. Simplicity, as described above, does not necessarily resolve ambiguities since two explanations can be equally simple. Here, we integrate the ideas of David Deutsch into our thoughts. He stated that "good explanations are hard to vary"[4] while bad explanations can easily be modified to meet the observed phenomena. With complex deformations we can easily produce similar mappings that can appear to be

[4]Talk by David Deutsch: "A new way to explain explanation", July 2009, Oxford, UK, http://www.ted.com/talks/david_deutsch_a_new_way_to_explain_explanation.html

symmetries. However, we cannot be certain that even one of them represents "real" symmetry due the ambiguous mappings. In general, we aim for solutions which are hard to vary. Consequently, a symmetry detection algorithm should reject all ambiguous symmetries or at least deal with the possibility that a symmetry is wrong or undesired. Rigid symmetries have such ambiguities mostly in the presence of slippable surfaces. But in contrast to more complex transformation classes, slippable surfaces have been investigated in detail by Gelfand and Guibas [2004] and can be integrated in the detection process.

In summary, rigid symmetries are an excellent choice for our purpose because they are defined by a small set of parameters, they can be interpreted as a physical model of the real world, and their ambiguities are well understood. More complex types of symmetries are possible, however, they should fulfill the discussed criteria of simplicity and hard variability, and ideally resemble a physical process. In this thesis, we will focus on rigid symmetries.

1.3 Overview and Contributions

This thesis is organized in two parts. In the first part, we investigate the potential use of geometric features for correspondence estimation and symmetry analysis. The second part builds upon the gained knowledge of the first part and derives potential modification operations for shapes based on partial symmetry. Please note that some figures and textual parts where taken from the individual publications mentioned in the following paragraphs.

Part I: Feature-based methods reduce the object to a set of features that is typically much smaller than the original object and perform the analysis on this reduced set. These methods have the advantage that they focus the attention on "interesting" regions of the objects and thereby perform faster than dense-sampling approaches. This requires that the desired correspondences or symmetries also exist in the feature representation. Here, we usually have a trade off between computational efficiency and recognition performance. We consider features as locally unique parts of the shape that are extraordinary in some respect. Features can be ,e.g., points, lines, regions. A popular choice is to use feature points because they are conceptually easy to use in correspondence problems: We only need to set feature points in correspondence. In Chapter 3, we present a new feature point detection technique for point cloud data [Bokeloh et al., 2008]. The key idea of this method is to identify areas that are strongly constrained with respect to the auto alignment problem. We perform this analysis on multiple scales and use local maxima as feature points. We demonstrate the applicability of the method on several example applications including symmetry detection [Berner et al., 2008] and isometric registration [Tevs et al., 2009].

While points as feature are conceptually easy and perform well in global correspondence problems, their potential is limited in case of partial symmetry detection. The reason for this is rooted in the enormous loss of information during

the feature detection stage. When a surface is reduced to a set of feature points (intuitively, we can think of them as corners), we will most likely lose important information ,e.g., surface parts between these points. Linear or planar structures are completely lost in this representation. In case of noisy scanner data with occlusions, we may lose further feature points that might be essential to detect the underlying symmetry. A much better feature representation can be achieved by using *lines* as features. In Chapter 4, we present a symmetry detection method that uses feature lines to find and refine potential symmetry transformations and to perform rapid geometric matching to score these transformations [Bokeloh et al., 2009]. This technique works on huge datasets, detects a large number of symmetries, and is able to decompose the shape into building blocks.

Part II: In the second part, we utilize symmetry information (that we gain with the methods described in the first part) to modify existing objects and synthesize new objects. Symmetry information offers more than partial correspondences or the knowledge that two things are similar. Symmetry also reveals context information: Suppose we discover object A two times in a scene and in the vicinity of the first instance we find an object B, however, not in the second instance. This means, we have observed B as an optional part in the context of A. Intuitively, it might be possible to copy object B to other places where we find object A. The key insight of this thesis is that the boundary between a symmetric region and a non-symmetric region can be used as a *docking site* where we can replace the existing non-symmetric geometry with its counterpart. In Chapter 5, we propose a method that systematically extracts these docking sites from a given shape and combines all docking sites and their associated shape operations into a shape grammar [Bokeloh et al., 2010]. Using this shape grammar, we can produce shape variations where we can provide strong guarantees for each resulting shape.

Many man-made shapes have dominant regular structures that are crucial for the overall appearance. Modifying these types of shapes, e.g., resizing, is either a time consuming task that requires sophisticated artistic and technical skills or results in unsatisfactory results (for example, with standard elastic deformation). In Chapter 6, we propose a method that automatically detects regular patterns and provides an interactive deformation technique that implicitly maintains regularities in order to preserve the shapes' appearance. In addition, we extend the concept of docking sites in order to change the repetition count of regular patterns without enforcing global cuts through the shape [Bokeloh et al., 2011].

In the following chapter, we give a short overview of related work and describe a couple of basic techniques that we will often use in this thesis. Then, we discuss the four main contributions:

- We introduce a generalized concept of geometric features that detects locally uniquely identifiable keypoints as centroids of area with locally minimal slippage (Chapter 3).
- We present a novel symmetry detection method that decomposes a shape into recurring parts and propose a technique for rapid geometric matching

(Chapter 4). Our method is currently state-of-the-art for building block decomposition of large-scale scanner data.

- We reveal a connection between partial symmetry and inverse procedural modeling that allows to synthesize similar shapes with strong guarantees (Chapter 5). In contrast to preexisting techniques, our method is able to derive a set of explicit, procedural shape operations directly from the input shape and combine this set of shape operations into a shape grammar without prior knowledge about the shape or the procedural rules.

- We propose a new content-aware deformation technique that uses regular patterns as underlying regularization principle and introduce the concept of sliding dockers in order to change the shapes' structure during deformation (Chapter 6).

2 Background

Do you realize if it weren't for Edison we'd be watching TV by candlelight?

Al Boliska

In this chapter, we will give a brief overview of related methods and describe basic techniques that we will use frequently within this thesis.

2.1 Related Work

2.1.1 Feature Detection

A feature is a part of some input data, e.g., corners or edges, that is locally unique in the sense that it is uniquely identifiable within a local neighborhood. In computer vision, feature detection has been investigated in the context of image matching and object recognition [Moravec, 1981; Harris and Stephens, 1988; Lowe, 1999, 2004]. In geometry, feature detection has first been examined for rigid registration of 3D shapes. Gelfand et al. [2005] base their technique on curvature on multiple scales and filter descriptors by preferring features that are more unique on each of the pieces to be registered. A drawback of their feature detection approach is that keypoints might not be well constrained along curves of constant mean curvature, so that random variations in feature positions might result. In addition, a local measure of curvature alone has only limited discriminative power. In contrast, Huber and Hebert choose a dense sampling approach: They compute a surface descriptor (spin-images) for every vertex on a uniformly sampled subset of the input meshes and use a RanSaC technique to find corresponding vertices with similar descriptors [Huber and Hebert, 2001]. The number of necessary samples can be reduced by referring points with specific surface properties like high curvature [Yamany and Farag, 2002] or bitangent points [Wyngaerd and Gool, 2002].

The method proposed by Li and Guskov [2005] adapts the concept of SIFT-Features [Lowe, 2004] to geometric data: They build a multi-scale representation of the input data and extract local extrema in the difference levels as features points. Huang et al. proposed a method for reassembling fractured objects using global geometric registration [Huang et al., 2006b]. They compute a set of surface signatures for every data point and apply a flooding based clustering algorithm to extract regions of similar signature. The centroids of the resulting surface patches are used as features, and the main axis determine the rotational alignment. The technique is performed on multiple scales to create a large number of potential features and descriptors. For feature matching, a series of filtering heuristics is used to exclude obvious mismatches, including ICP-based alignment as final step. Global verification of the alignment is done with statistical forward search. The drawback of the approach is that the region growing based feature extraction is likely to produce a large number of both false positive and false negative matches, as the connectivity of the surface patches can easily change, leading to non-matching features. In their target application, this is not an issue, as only a small number of final matches are necessary and the forward search based global verification is able to compensate for this drawback easily.

Other types of features have been investigated for 3D shapes: Gumhold et al. [2001] and Pauly et al. [2003] extract crease lines from point cloud data via PCA analysis of local neighborhoods. Slippage analysis can be used to segment a shape into regions with equivalent slippage directions [Gelfand and Guibas, 2004] and thereby form region features. Similarly, Schnabel et al. [2007] extract shape primitives from point cloud data using a RanSaC approach.

Recently proposed methods try to learn geometric features: Sunkel et al. [2011] learn geometric properties and spatial configurations of a set of user-marked features using a Markov chain model in order to detect semantically similar instances with varying geometry. Kalogerakis et al. [2010] learn an objective function from a collection of labeled examples and use this function for segmentation and labeling of 3D meshes.

Geometry descriptors: Descriptors, also referred to as *signatures*, describe the local geometric properties of a feature. They characterize a feature and are used to find potentially corresponding features. Ideally, a descriptor is invariant under rotation so that a global rotation applied to an object does not change the descriptor of a feature. This invariance is important for global registration techniques where no assumptions are made on position or orientation of objects. There are a large number of techniques to describe geometry with a small digest of data. Many of these matching techniques have been designed for efficient object retrieval from large data bases. Thus, in order to perform a large number of comparisons efficiently, these methods typically employ mappings of geometric objects to small vectors of characteristic numbers: Kazhdan et al. [2003b] employ power spectra of a spherical harmonics description of the geometry to compute a rotationally invariant geometry descriptor. Novotni and Klein [2003] extend this approach to Zernike polynomials to reduce the invariant space that maps to the same descriptor. Gatzke

et al. [2005] consider maps of local curvature values. Another standard technique are spin images [Johnson and Hebert, 1999; Huber and Hebert, 2001] that compute an average trace of geometry intersecting a plane rotating around the surface normal. A related approach based on spherical harmonics has been proposed by Frome et al. [2004]. Mitra et al. [2006a] propose a matching technique based on min-hashing that also allows for partial object matching.

In Chapter 3, we introduce a generalized concept of keypoint detection. Our method considers features as centroids of surface area with minimal slippage. Further, we describe a feature line extraction method for 3D shapes in Chapter 4.

2.1.2 Symmetry Analysis

Early attempts in symmetry detection focused on symmetries in small point sets [Atallah, 1985; Wolter et al., 1985; Alt et al., 1988]. These methods were rather restricted and not practical for imperfect symmetries in larger 3D data sets. A class of approaches that enable wider applicability ,e.g., approximate symmetries, is known as transformation voting. The basic idea is to create candidate transformations from potentially corresponding sample points and insert these candidate transformations into the transformation space. Dominant symmetries will most likely form a large clusters in this space. Transformation voting techniques for symmetry detection in geometry have been introduced by Mitra et al. [2006b], and Podolak et al. [2006], as well as Loy and Eklundh [2006] for image features. While the two latter are restricted to reflections, translational and rotational symmetries, respectively, the technique of Mitra et al. considers the full range of similarity transformations. The symmetry detection technique of Mitra et al. has subsequently been extended to automatic object symmetrization [Mitra et al., 2007]. Pauly et al. [2008] proposed an extension to find regular patterns: If the objects show regular patterns of symmetries, such as regularly spaced rows of windows in a building, one can explicitly look for these patterns in transformation space to obtain much more stable results. However, this approach can only identify such patterns; symmetric parts in isolated instances or as members of different patterns are not identified as belonging to the same class of objects.

Gal and Cohen-Or [2006] propose a variant of transformation voting that uses geometric hashing [Lamdan and Wolfson, 1988] of salient features. The authors give examples of detecting a small number of symmetric parts within an object. Simari et al. [2006] detect planar reflective symmetries by computing an auto-alignment of parts of a shape with itself using iteratively reweighted least-squares. This yields a nested symmetry decomposition. Such approaches are limited to cases where large and small scale symmetry patterns correlate. Martinet et al. [2006] propose a technique that uses a transformation to generalized moment functions in order to compute global symmetries of 3D shapes. Kazhdan et al. [2003a] analyze objects for central symmetry and use this as a descriptor for shape retrieval. Schnabel et al. [2008] use graphs of fitted geometry primitives to perform object recognition. Lipman et al. [2010] introduce symmetry factored embedding as a

new tool for shape analysis. A very interesting application of symmetry detection is shape completion: Thrun and Wegbreit [2005] compute symmetries of partially scanned objects to complement the partially shape. In a recent work, Zheng et al. [2010] extended this idea to reconstruction of 3D urban scenes by exploiting regular patterns.

Recently, many works focus on detection of intrinsic symmetries. Different techniques were proposed to find intrinsic reflective symmetries [Ovsjanikov et al., 2008; Xu et al., 2009a; Raviv et al., 2007, 2009; Kim et al., 2010; Raviv et al., 2010]. Mitra et al. [2010] proposed a method for intrinsic regularity detection: They use multidimensional scaling to embed the shape in 2d domain where 2d grids appear as Euclidian grids. Ben-Chen et al. [2010] investigated on detection of intrinsic continuous symmetries also known as Killing vector fields (KVF), named after Wilhelm Killing. They formulate KVFs as a variational problem and thereby allow to find approximate KVFs. In a recent work, Berner et al. [2011] introduced a new concept of subspace symmetries. They assume that symmetric elements can be characterized by a low dimensional shape space and thus generalize the notion of symmetry to general deformations.

We will present a new symmetry detection method based on feature lines in Chapter 4 that exploits local coherence in symmetries. Our approach is not restricted to hierarchical symmetries or regular patterns which is crucial for transformation voting techniques as [Mitra et al., 2006b; Pauly et al., 2008] in order to process large data sets.

2.1.3 Texture and Geometry Synthesis

Non-parametric texture synthesis deals with creation of textures from exemplar images, for example photographs. This thesis, especially part II, is motivated by non-parametric texture synthesis [Efros and Leung, 1999; Hertzmann et al., 2001; Kwatra et al., 2003], which optimizes for similarity of local, overlapping neighborhoods to corresponding regions in the exemplar image. Being formulated as an optimization problem, this leads to a hard Markov random field (MRF) inference problem. Texture synthesis has also been applied to 2D vector graphics [Barla et al., 2006; Ijiri et al., 2008] and the notion of local r-similarity of neighborhoods (see Chapter 5) has been generalized to 3D geometry by [Rustamov, 2008].

Texture synthesis can be applied to synthesize 3D geometry by discretizing a base surface and synthesizing details on top of it [Lai et al., 2005; Nguyen et al., 2005; Chen and Meng, 2009; Zhou et al., 2006; Zelinka and Garland, 2006]. This requires that the coarse scale base geometry is given as user input. Alternatively, one can discretize the ambient space itself, synthesizing occupancy in space. Techniques include voxel models [Bhat et al., 2004] and implicit functions [Sharf et al., 2004; Lagae et al., 2005]. However, it is difficult to find good solutions based on heuristic MRF optimization; creating closed and well defined geometry is more challenging than synthesizing plausible 2D images. None of the known methods have so far demonstrated results where large scale models with complex structure,

such as complete buildings, are synthesized from scratch. Merrell [2007] propose a related algorithm that expects building blocks aligned with a regular grid as input. These are then placed automatically with consistency across grid faces, again involving a discrete MRF labeling problem. The technique can handle arbitrary boundary conditions but, unlike our approach, building blocks are required as input rather than output and the regular grid structures limits the design space significantly. This has been addressed in [Merrell and Manocha, 2008], where cells are formed by intersecting planes through faces of the exemplar model, implicitly creating the grid structure. However, the approach is limited to very low complexity input exemplars (examples in the paper have up to 39 input faces). Cabral et al. [2009] examine a related idea: User specified building blocks and a connectivity graph are the input and the system then optimizes vertex positions and textures to form a closed model. It requires the building blocks and their interconnection rules as input.

2.1.4 Shape Grammars and Inverse Procedural Modeling

Grammar-based modeling of geometry and texture is one of the most successful procedural modeling paradigms. Applications include plant modeling [Prusinkiewicz and Lindenmayer, 1990] and modeling of cities [Parish and Müller, 2001] and buildings [Wonka et al., 2003; Müller et al., 2006]. Talton et al. [2011] present an approach to control procedural modeling by performing maximum a posteriori estimation in order to generate objects that meet complex boundary constraints.

Inverse procedural modeling is referring to the reverse process, where rules (such as shape grammars) have to be derived from example geometry. This goes beyond geometry synthesis in the sense that it not just creates similar models from exemplars but also describes the structure of the space of such models. An early approach is the work of Hart and Flynn [1997] who derive fractal branching rules for L-systems from 2D example graphics by geometric hashing, however, being limited to simple L-systems with a few rules only. As described above, Pauly et al. [2008] proposed a method for detection of regular patterns from example geometry. In [Mitra and Pauly, 2008], this technique is used for creating variants from example models by changing the replication frequency or editing symmetric pieces simultaneously. [Yeh and Měch, 2009] analyze 2D vector graphics to detect complex 1D patterns along curves with secondary structure. Very recently, [Šťava et al., 2010] extend this idea to detect hierarchies of patterns, yielding an L-system describing the example geometry, similar to the context free representation that will present in Chapter 5. Our scenario is more complicated as it deals with 3D surfaces that need to be assembled in a consistent way, without holes in the surface. In contrast, 2D vector graphics can just be composited arbitrarily. A lot of work has been presented that uses procedural rules to fit geometry to image input rather than geometry [Aliaga et al., 2007; Müller et al., 2007; Neubert et al., 2007; Tan et al., 2007; Xiao et al., 2009], which is a very hard inverse problem. These techniques use predefined classes of rules (such as hierarchical regular subdivisions of

facades [Müller et al., 2007]) or a significant amount of user input to facilitate image interpretation but do not attempt to create shape grammars automatically from scratch.

A number of approaches have been developed for recombining shapes out of parts. A number of approaches that utilize manual part composition have been described [Funkhouser et al., 2004; Pauly et al., 2005; Kraevoy et al., 2007], as well as automatic methods for detecting structural regularity [Pauly et al., 2008]. The recent method of Wang et al. [Wang et al., 2011] infers a scene graph structure for an unannotated 3D mesh to allow for both continuous and discrete parameter variations.

Merrell et al. [2010] propose a procedural technique to generate building layouts using a Bayesian network trained on examples. Recent methods try to suggest arrangements or components to the user [Chaudhuri and Koltun, 2010; Merrell et al., 2011; Chaudhuri et al., 2011; Fisher et al., 2011].

In Chapter 5, we present a novel approach to inverse procedural modeling of 3D shapes. Given an input exemplar, we derive explicit shape modification rules that guarantee to produce shapes similar to the exemplar. In contrast to previous exemplar-based methods, our approach is fully automatic and not restricted to a specific structure of the input.

2.1.5 Shape Deformation

Free-form deformation has a long tradition in computer graphics. Early techniques use smooth basis functions to interpolate between control points [Sederberg and Parry, 1986; Coquillart, 1990]. Recent work constructs basis functions specific to a set of control points or a control cage [Ju et al., 2005; Joshi et al., 2007; Lipman et al., 2008; Ben-Chen et al., 2009]. Many of the techniques are based on variational calculus: Local regularizers are traded off against the user's constraints. The regularizers aim to maintain local similarity to the input. Elastic deformation models [Terzopoulos et al., 1987], which minimize the non-rigidity of the deformation, are particularly popular [Botsch and Sorkine, 2008]. Variants of regularizers include volume preservation [von Funck et al., 2006], plastic deformation [Mezger et al., 2008], similarity transforms [Liu et al., 2008], and thin-plate splines [Allen et al., 2003]. We use elastic deformation as a "base regularizer" to diffuse stretch and to preserve geometry for which no structural information could be inferred. Our implementation adopts the technique of Sorkine and Alexa [Sorkine and Alexa, 2007] that preserves co-rotated distance vectors in a least-squares sense. We extend this to a volumetric subspace formulation [Zhou et al., 2005; Huang et al., 2006a; Sumner et al., 2007; Adams et al., 2008]. This allows interactive handling of large meshes and provides robustness against unfavorable mesh topology so that we can handle arbitrary "triangle soup."

Local regularizers do not recognize higher-level structural properties in the shape. Consequently, these techniques still expose a large number of degrees of freedom to the user, who has to manually ensure that important structural proper-

ties are maintained. This is acceptable for many organic shapes such as creatures, but highly structured objects with regularities at multiple levels of detail, such as many man-made objects, are difficult to handle. Kraevoy et al. [Kraevoy et al., 2008] use an elastic-type model that adapts to the vulnerability of the local content. In addition, three global stretch axes are fixed, which avoids bending artifacts but also limits the applicability of the technique to axis-aligned stretching.

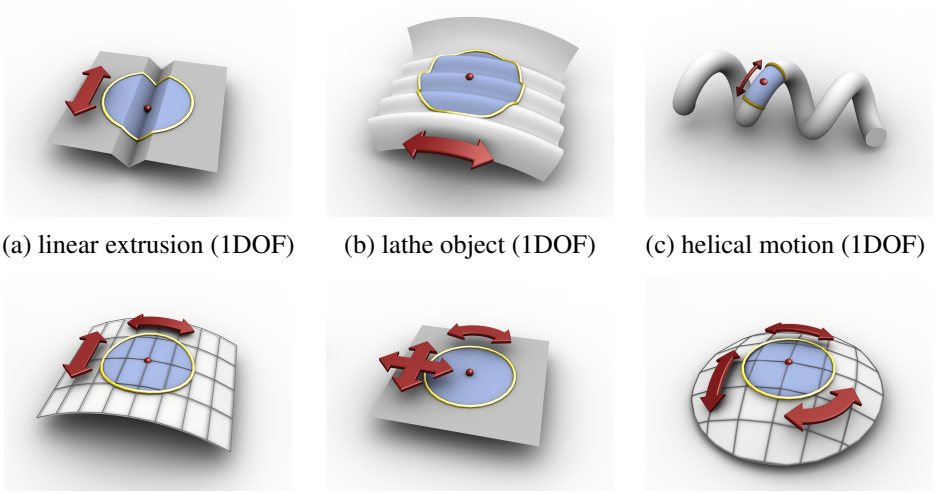
Xu et al. [Xu et al., 2009b] introduce slippage analysis for free-form shape deformation, using it to construct a joint-aware deformation model. We also use slippage analysis, but employ it to discover continuous symmetries that are used to maintain the pattern structure of the input.

The influential iWires system [Gal et al., 2009] maintains global structural properties of the shape by building constraints that preserve similarity of symmetric parts [Mitra et al., 2006b; Podolak et al., 2006; Simari et al., 2006], as well as parallelity and orthogonality of salient feature lines. Huang et al. [Huang et al., 2009] use similar ideas for 2D vector graphics, and Zheng et al. [Zheng et al., 2011] propagate editing operations based on similarity of components. Using such global knowledge greatly facilitates shape editing, but a key limitation remains: The deformation function is still a continuous, bijective map between input and output. This does not allow the insertion or removal of elements, which can be desirable in response to significant stretch.

In Chapter 6, we present a pattern-aware deformation approach that overcomes many limitations of previous structure-aware methods. Our method is based on regular patterns and tries to preserve their structure in the deformation process ,i.e., we are not restricted to three fixed scale axes such as in [Kraevoy et al., 2008]. Further, we use discrete regular patterns to relax stretch by inserting and deleting elements which has not been shown before.

2.2 Slippage Analysis

Slippage analysis poses the problem whether a rigid matching of a piece of surface with itself is well constrained. Conceptually, a point-to-plane ICP error function, dependent on 3 rotation and 3 translation variables, is setup for a configuration where the surface is perfectly aligned with itself. Then, the Hessian matrix of this objective function reveals directions in which the problem is not well constraint: The eigenvectors of the zero eigenvalues of this matrix correspond to transformations that map the surface onto itself in an infinitesimally sense (instantaneous motion). For example, for a cylinder mapping to itself, there are one rotational and one translational degree of freedom, resulting in two eigenvectors with zero eigenvalue (see Figure 2.1). The cylinder can be rotated around its rotational symmetry axis and shifted along this axis without changing its shape locally. In general, non-zero eigenvalues are a necessary condition for the auto-alignment problem being well defined. For non-perfect data sets, such as scanned point clouds, we cannot expect zero eigenvalues but small eigenvalues indicate an underconstraint auto-alignment.



(a) linear extrusion (1DOF) (b) lathe object (1DOF) (c) helical motion (1DOF)

(d) cylinder (2DOF) (e) planes (3DOF) (f) spheres (3DOF)

Figure 2.1: *Slippable surfaces with respect to rigid motions.*

As shown in [Gelfand and Guibas, 2004], the hessian matrix of the alignment problem is given by:

$$C = \sum_{i=1}^{n} \begin{pmatrix} c_i c_i^\top & c_i n_i^\top \\ n_i c_i^\top & n_i n_i^\top \end{pmatrix} \tag{2.1}$$

where $c_i = (p_i \times n_i)$ and $C \in \mathbb{R}^6 \times \mathbb{R}^6$. p_i refers to the i-th surface point of a surface discretization and n_i refers to its normal. Please note that as slippage is derived from a point-to-plane matching, normal information is necessary to setup the Hessian matrix.

2.3 RanSaC

RANdom **SA**mple **C**onsensus (short RanSaC) is an excellent tool for model fitting to noisy data with a large number of outliers and we will use and adapt this paradigm to symmetry detection in the following chapters. Here, we will describe the basic principle of RanSaC with the example of line fitting to a set of points as presented in the original paper [Fischler and Bolles, 1981]. RanSaC generates many hypotheses (here: lines) by randomly sampling the data, assigns a score to each hypothesis and keeps the hypothesis with the highest score. A key concept in RanSaC is how hypotheses are created: In the line fitting example, a line can be parameterized by two points. Therefore, a candidate line can be easily generated by selecting two random data points. If a line is present in the data and supported by a sufficiently large number of points, the probability is high that a candidate line is generated that is close to the optimal line. Scoring is usually done by counting all points within a certain distance to the model (outlier distance).

Part I

Feature-based Symmetry Analysis

Symmetry analysis and other correspondence problems in geometry suffer from a relatively large search space. For rigid symmetries this means we have a 6D search space which already confronts us with a computational problem considering that verifying a single transformation already has linear worst case complexity. Besides the computational cost of a brute force search it is also not necessary to investigate the full search space; in many cases only a fraction of the space holds meaningful rigid symmetries.

Feature-based methods present a simple but elegant way to reduce the search space while maintaining important structural information. Ideally, all symmetries can be observed in the feature representation. The key idea is to represent a shape as a compact set of features and guess potential symmetry transformations by setting features into correspondence. This way the search space is reduced to transformations that already have some initial proof which justifies a precise investigation. Further, the feature representation can be used for quick verification.

In Chapter 3, we will present a novel feature detection technique for point features and show applications to registration problems and symmetry detection. Furthermore, we will introduce a new symmetry technique for rigid symmetries based on line features in Chapter 4.

3 Feature Points for Correspondence Analysis

You know those balls that they put on car antennas so you can find them in the parking lot? Those should be on every car!

Homer J. Simpson

Feature detection techniques aim at finding locally extraordinary surface elements that are invariant to global transformations and noise in the surface. This invariance is important for global registration problems where relative transformations between shapes are unknown. If we are able to identify the same features on different shapes, we can derive a transformation between both shapes. A feature point, often referred to as keypoint, is usually defined by its local neighborhood; typically an Euclidian or an instrinic sphere. The task of a keypoint detector is to find spheres that encapsulate locally unique geometry; here, it is most important that this "uniqueness" is robust against global transformations and noise. Since this is not an easy task in general, detection methods usually derive noise-resistant, rotationally-invariant properties of the local geometry instead of using the geometry directly. Additionally, a keypoint detector should also reduce the amount of information in a shape while still returning a characteristic subset.

An important problem for feature-based techniques is the ability to *reliably* detect a sufficient number of keypoints: For traditional applications such as performing rigid registration of piecewise scanned point clouds [Huang et al., 2006b; Mitra et al., 2006a; Gelfand et al., 2005; Li and Guskov, 2005; Li et al., 2006; Huber and Hebert, 2001], a small number of correctly established correspondences is sufficient, as a rigid motion is uniquely determined by specifying point-to-point correspondences between two triples of non-collinear points. However, some application areas of geometry processing that recently gained importance require many more correspondences: For example, deformable alignment of surfaces [Wand et al., 2007; Brown and Rusinkiewicz, 2007] has many more degrees of freedom so that a larger number of correspondences, covering all parts of the object, is necessary to make the alignment problem globally well-defined. In our experience, this

can be a serious issue for traditional keypoint detection techniques. In the context of partial symmetry detection, we require a set of keypoints for each recurring part located at the same positions in each instance. Again, for such an application, a much larger number of reliable keypoints is necessary. Most established feature detection techniques detect keypoints at points of maximum curvature [Lowe, 2004; Gelfand et al., 2005; Li and Guskov, 2005; Yamany and Farag, 2002; Wyngaerd and Gool, 2002]. For the aforementioned application areas, this notion is often not general enough so that an insufficient number of keypoints is extracted for many input models.

We want to to generalize the notion of keypoints to include as many locally uniquely localizable regions as possible without compromising stability and reliability of detection. The basic idea of our technique is that a feature must have the property that within a local neighborhood the position of the keypoint must be uniquely defined so that the notion of feature correspondences is locally well defined. In addition, for many applications it is also useful to have a well-defined rotational alignment in order to establish correspondences with corresponding coordinate frames rather than just point-to-point correspondences. This means that the rotational degrees of freedom should also be well defined for the feature within its local neighborhood.

In order to determine such points we examine the local geometry at each surface point contained within a ball of fixed radius. The radius determines the scale at which the feature detection takes place. In order to detect regions with well defined translation and rotation, we employ the *slippage analysis* of Gelfand and Guibas [2004] to compute a measure of how well constraint an auto-alignment problem of the feature regions is. Keypoints are then defined as local maxima of this constraintness measure. We extract these maxima using a meanshift clustering algorithm and verify the stability by filtering out points where the maxima are not pronounced enough to allow a stable detection. This criterion for determining keypoints is more general than just looking for local maxima in curvature of the geometry itself; a piece of geometry containing a complex geometric pattern might be strongly constrained in its auto-alignment although it might appear flat in average, not providing a local extremum in curvature. As the local well-constraintness of the alignment in translation (and, depending on the application, also rotation) is a necessary criterion for a geometrically defined feature, our approach also defines a natural and very general class of features. In order to eliminate the free radius parameter of the detection radius, we generalize our technique to multi-scale feature detection and detect maxima in scale space, adopting the conceptual framework of SIFT keypoint detection [Lowe, 2004].

The novel notion of general keypoint detection based on a slippage analysis is the key idea and main contribution of this chapter. We complement our technique by adding a similarly general descriptor technique: In order to compute descriptors for the obtained features, we employ a two stage filtering approach: First, we compute local curvature histograms of the feature area [Gatzke et al., 2005] to obtain a filtering criterion to rapidly reject unlikely matches. Afterwards, for the smaller set

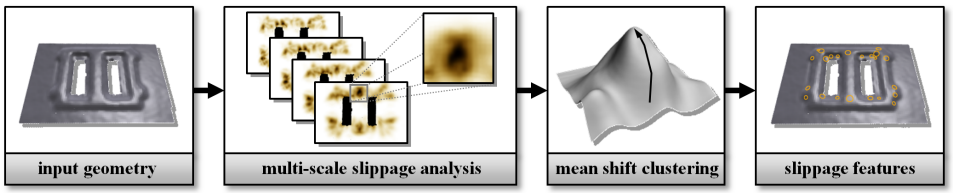


Figure 3.1: *Overview of slippage feature extraction: First, we perform multi-scale slippage analysis on the input model. Then, we extract local maxima of slippage by mean-shift-clustering.*

of remaining matches, we employ a local iterated closest points (ICP) alignment to compute a more precise matching score [Huang et al., 2006b]. In order to make this alignment computation fast and reliable, we employ a cross-correlation technique based on the fast Fourier transform (FFT) in order to compute a good initial rotational alignment.

We demonstrate the benefits of our technique by examining example applications in both rigid and non-rigid alignment, and also examine the potential for symmetry detection based on generalized keypoints. In addition, we provide a thorough empirical analysis of stability and matching precision of the proposed technique in comparison to related work.

3.1 Slippage Keypoints

In order to apply slippage analysis for our purpose, we first discretize the surface into a cloud of points. In addition, we also have to compute surface normals. If our input is given as a triangle mesh, we start by computing a dense point cloud by a uniform random sampling of the surface. Next, for both triangle and point cloud input, we continue with a Poisson-disc resampling step that results in a uniform, well distributed sampling of the surface [Wand and Straßer, 2002]. The position of the resampled points is projected back on a quadratic moving least squares surface fit of the original data points to optimize the representation accuracy. After this computation, a uniform surface sampling is obtained with known sampling spacing. In case of triangle meshes, we retain the original triangle normals as initial normals. For point clouds, we compute normal information by a PCA analysis [Hoppe et al., 1992]. We weight the contributions of neighboring points with a Gaussian windowing function with a standard deviation of 1.5 times the median point spacing.

Subsequently, we go through each point of our surface discretization and compute the local slippage analysis of a spherical neighborhood of points of a fixed radius 2σ. For correctly band limiting the computation, we add a Gaussian weighting window with standard deviation σ to each term of each surface point in Equation (2.1), i.e. the neighborhood radius is chosen to truncate this Gaussian window as soon as the weighting function becomes negligibly small. Before slippage compu-

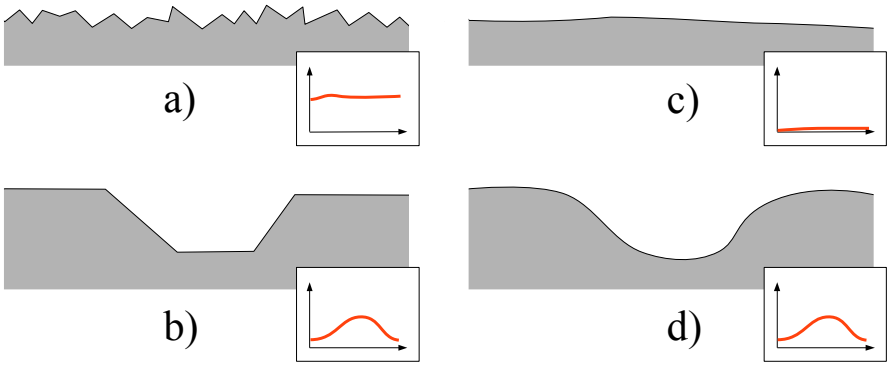


Figure 3.2: *The normal smoothing problem (the small graphs show the slippage values schematically): Fine scale geometry variation can lead to low slippage values due to varying normals (a); at coarser scales, this is undesirable. Smoothing the normals resolves this issue (c). For large scale features (b), the normal smoothing will still preserve the detected feature (d).*

tation, we normalize the extents of the region considered to a unit bounding box centered at the origin. This makes sure that rotational and translational degrees of freedom are always measured with the same relative weighting. From the eigenvalue decomposition, we consider only the smallest eigenvalue λ_6 as a measure for constraintness. In order to make the measure invariant under the amount of local area gathered, we additionally divide by the largest eigenvalue λ_1 which is guaranteed to be non-zero (any non-empty surface is at least translationally constraint in one normal direction). A low value of this ratio indicates slippable motion in at least one direction; a high value indicates a more constraint rigid motion. Please note that using the smallest eigenvalue of the Hessian (2.1) means that we will exclude keypoints that are not rotationally well constrained, such as rotationally symmetric bumps. It is straightforward to setup a similar slippage analysis problem that ignores rotational degrees of freedom altogether in order to extract translationally invariant keypoints only. For clarity of exposition, we restrict ourselves to the case of rotationally well-constraint keypoints in this work. For the rest of our pipeline, we stick to this scenario and exclude features that are rotationally ambiguous. Filtering out points that are slippable under infinitesimal rotations is thus the first step, as this violates a necessary condition for unique matching.

3.1.1 Multi-scale Analysis

Next, we consider the problem of estimating slippage values for multiple scales. At this point, we are facing the problem of coupling the normal variation to the geometric level of detail: In a multi-scale analysis, our objective is to examine geometric features in multiple frequency bands. For a coarse scale analysis, small scale features should not be taken into account. However, the slippage analysis

is formulated as constraining rigid motion by aligning planes defined by points and normals. Therefore, small scale variations in the normals can make the auto-alignment strongly constrained although this is not warranted at the frequency level considered. The problem is illustrated in Figure 3.2: The plane in Figure (a) is non-slippable due to normal variations at a much smaller scale. In order to couple the normals to the scale of considerations, we have to band-limit the frequency of normal variation accordingly (Figure 3.2c). We do this by filtering the normals with a Gaussian low-pass filter of support proportional to the size of the region considered. Small scale variations will be removed, while large scale geometric features will still be retained (3.2 b,d). In experiments, it turns out that the best results are obtained from a rather aggressive filtering strategy, where only relatively low frequency normal variations are retained at each scale (we use $\sigma_{smooth} = 1.25\sigma$). Please note that we do not need to filter the geometry in a similar way, as point positions with small scale noise will still produce realisticly slippable results as long as the normals are smoothed correctly. Another way to formulate this fact is that normals and geometric positions are measured in different units, where normals are not affected by scaling while positions are.

Repeated subsampling, normal filtering and slippage evaluation will yield a sequence of scales with support $\sigma_i = \sigma_0 F^i$ ($F > 1$). In the following, we denote $\theta_{i,s}$ the slippage indicator resulting from the local slippage analysis computed at point x_i with support σ_i. In the scale-space domain we now search for local maxima. A natural technique for extracting this information in a robust way in the presence of noise, is mean shift clustering [Comaniciu and Meer, 2002]. This technique first convolves the data points with a low-pass filter kernel and then performs gradient ascend to find maxima. All points ascending to the same maximum are comprised in the same cluster, and the location of the maximum becomes our representative key point. In our case, we employ a standard Gaussian windowing function as mean shift kernel and set the support radius to a fixed value of 2 times the median sampling spacing. In scale direction, we use a Gaussian window of standard deviation 0.5, where 1.0 corresponds to one scale level F. As we are only interested in the location of the maxima, we can optimize the computation: We first compute candidate points at discrete positions that are local maxima on each scale after filtering in the spatial domain only. Afterwards, we start a gradient ascend in scale-space to find final maxima at continuous spatial position and continuous scale values. Doublets and diverging points are removed at that stage. In a final step, we evaluate how well determined the extracted maxima are. We estimate the second order derivatives by finite differencing and delete points with small curvature (below a threshold H_{min}) in the slippage value, as these points will not be detected reliably under the influence of noise and small scale random variations of geometry.

3.2 Descriptors and Feature Matching

Having determined a set of keypoints, we need to compare the local geometric neighborhood in order to determine candidate correspondences. In our case, the local neighborhood always corresponds to the geometry contained within a fixed distance of 10-15 $\times$ the median point distance. In order to match the generality of the keypoint detector, our feature comparison function is based on a direct geometric comparison. We proceed in three steps: Signature based pruning, rotational alignment, and geometric alignment.

3.2.1 Signature-based Pruning

As a quick test to reduce the number of matching candidates, and thus reduce the number of the later, more expensive alignment steps, we compute local feature signatures. The main idea is to compute histograms of surface attributes over concentric rings around the keypoint. By construction, such a signature is rotationally invariant. As attribute value, we will use mean curvature; it is straightforward to generalize this technique in order to use additional attributes such as surface color, if available.

Let us assume, that we have an attribute a_i defined for every point x_i of the data set that is invariant under rotation and translation, such as mean curvature. Then, a pair of corresponding features should have roughly the same distributions of this attribute in their environments. Given a feature point f, we denote a ring histogram H_r as a histogram of the attributes a_i weighted with a spatial function ω_r:

$$H_r(f) = \{H_{r_b}(f) | b \in \{1, \ldots, N_b\}\} \tag{3.1}$$
$$H_{r_b}(f) = \sum_{i \in I} B_b(a_i)\omega_r(\|x_i - f\|)$$

A ring histogram contains N_b bins. B_b is a function which returns 1, if the given attribute a_i falls into bin b (0 otherwise). We can assume, that the Euclidean distance between f and x_i is roughly equal for corresponding features. We take this into account by computing multiple histograms with different spatial support:

$$\omega_r(x) = max(0, \lambda_h \, |r - x|) \tag{3.2}$$

For stability reasons, we smooth the histograms with a triangle filter. The signatures are invariant under rigid transformations, easy to compute and robust to spatial noise. Here, we use the mean curvature as attribute: For every point in the data set, we precompute an approximation of the curvature by fitting a quadratic patch onto the neighboring data points. We use the computed normal to define a local tangent coordinate system and compute a weighted least-squares fit of a bivariate quadratic polynomial. From the resulting coefficients, we compute the mean curvature analytically. In order to compare two signatures, we just compare the Euclidian distance of the scalar vectors formed by the corresponding histograms

and reject the match if a certain threshold is exceeded. Implicitly, this criterion will put larger emphasize on the outer rings as the histogram entries will consist of larger numbers due to the larger number of neighboring points counted. This is desirable, because more area is covered by these rings.

3.2.2 Local Feature Alignment

Having rejected most incompatible correspondences by comparing the signature vectors we can still expect to retain a small number of false positive matches, as the histogram criterion is not injective. Therefore we proceed by a geometric registration of the features and compare the residual of the match.

Rotational alignment: The first problem we are facing is a correct rotational alignment of the features. So far, only the translation is known due to the keypoints, which is not sufficient to guarantee the convergence of the later ICP alignment step. Huang et al. address this problem by computing a principal component analysis from a feature cluster and used the resulting eigenvectors for alignment [Huang et al., 2006b]. Another approach is to use the principal frame defined be the normal and the directions of maximal and minimal curvature [Li et al., 2006; Mitra et al., 2006b], and drop features with ambiguous principal frames. Both techniques have the drawback that they might return no well-defined result in rotationally symmetric situations, even though the auto-alignment problem is still well defined. In order to improve upon this, we employ a spectral cross-correlation technique: We assign a orthogonal local frame $F_i = (u_i, v_i, n_i)$ to a feature f_i, where n_i is the feature normal and u_i, v_i are arbitrary orthogonal tangents. Then we transform the local environment into this frame and consider the surface as 2D height field. When two features match then there exists at least one rotation angle α_t that maximizes the scalar product

$$\arg\max_{\alpha_t} \langle H_1, R_{\alpha_t}(H_2) \rangle \tag{3.3}$$

where H_1 and H_2 are height fields and $R_{\alpha_t}(H)$ is a transformation which rotates the height field H by α around the normal. Since we have only a sparse set of surface points, we need to produce a dense representation for comparison. For this purpose, we use the *splat-pull-push* method proposed by [Gortler et al., 1996] to create a regularly sampled array of height values. After resampling, we perform a polar coordinate transformation of the values, i.e. we resample the values into an additional array that is indexed by angle and distance to the origin. For each ring of constant distance, the cross correlation function defined above corresponds over the angle of rotation to the 1D convolution of the two signal vectors. As we are looking for the maximum of this value over all rings, we sum up the 1D convolution functions. Each such convolution can be computed efficiently in the Fourier domain [Castro and Morandi, 1987]: For each feature, we precompute a fast Fourier transform of its height field values of constant distance, obtained from the polar coordinates transform. Then, a point-wise multiplication of the

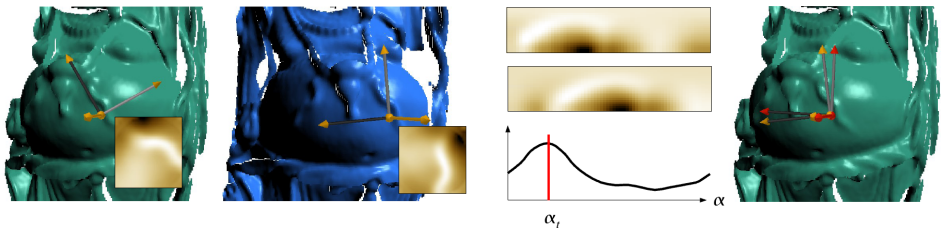


Figure 3.3: *Approximate alignment of two features from different views of the buddha data set: We transform the parametric images (height fields) of the feature environments into the polar domain, use the phase correlation technique to find a good initial transformation (red coordinate frame) and perform local ICP to refine the transformation (yellow).*

Fourier coefficients yields the convolution. Each rotational alignment can now be performed in $O(n)$ time, where n is the number of entries of the transformed height field. Each feature needs $O(nlogn)$ precomputation. This is much more efficient than performing an $O(n^2)$ brute force test for each of the potentially quadratic number of candidate feature matches. For subpixel accuracy, we compute a local quadratic fit and refine the peak of the rotational correlation function. Having found one or more peaks, we compute the transformation between the two features with respect to the estimated angle. For most comparison purpose it is sufficient to take the biggest peak only. Some features however have multiple peaks due to similar structures. We reject all peaks that are less than 0.8 times the maximum peak. In case we still find multiple rotational matching candidates, we dismiss the keypoint altogether, as it can be considered to be not rotationally well constrained. Alternatively, it is also possible to keep and test multiple alignments, if rotationally uniqueness is not required for the target application.

Geometric alignment (ICP): With the estimated approximate transformations, we perform local point-to-plane ICP alignment [Chen and Medioni, 1992] and take the minimum alignment residual (average of the point to plane distances) as feature distance. If the ICP alignment does not converge, we reject the correspondence pair. Figure 3.3 shows an approximate alignment result with two corresponding features on the buddha data set.

3.3 Applications

In order to evaluate the practical performance of the proposed feature detection scheme, we have implemented three prototypical example applications: Global rigid registration, global deformable registration, and a simple feature based symmetry browser.

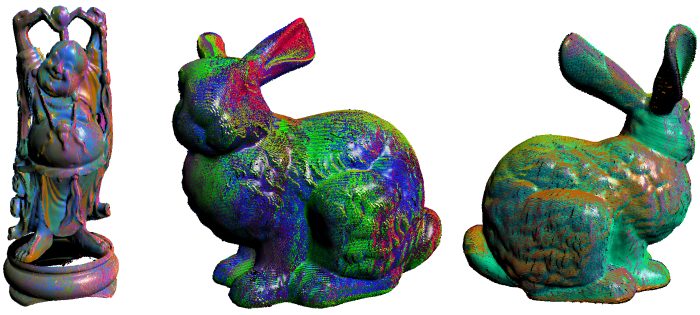


Figure 3.4: *Results of the registration technique for the case of global rigid alignment. Each individual view has a different color. The results where obtained from corresponding slippage features only without performing a final ICP step.*

3.3.1 Global Rigid Registration

For global rigid registration, we first compute keypoints and matching features as described above. After that, we use a global validation algorithm to extract a consistent set. We employ the spectral graph matching technique of Leordeanu and Hebert [Leordeanu and Hebert, 2005]. The algorithm considers pairs of correspondences and assigns a plausibility score to these pairs based on the difference in Euclidian distance on source and target shape. This yields a compatibility matrix. The diagonal elements of this matrix refer to compatibility of a correspondence with itself; we insert normalized residuals from the ICP matching stage in order to evaluate the plausibility of the correspondence itself. Normalization and mapping of distance discrepancies are handled exactly as described in the original paper [Leordeanu and Hebert, 2005]. Next, the eigenvector with the largest eigenvalue is computed using a power iteration. Large entries in this vector represents correspondences that are compatible to a large set of other correspondences, while outlier correspondences receive low scores. A final quantization step (again following closely the original paper), sets the eigenvector entries to zero and one, deleting incompatible matches for which the Euclidian distance differs by more than a fixed threshold. Lastly, we compute a global rigid alignment from the validated correspondences using standard SVD-based matrix decomposition. We show the alignment results *without* a final ICP alignment in order accurately depict the obtained registration accuracy from feature matching only.

3.3.2 Deformable Registration

Our deformable matching application uses exactly the same algorithm as the rigid case, just substituting Euclidian distances by geodesic distances along the surface, as we expect the deformation to be at least approximately isometric [Anguelov et al., 2004]. Geodesic distances are computed in two steps: First, we form a

Data	points	scales	Smoothing	Slippage	Meanshift	Total
human	89147	10	20	80	31	131
bunny	40256	10	10	30	9	49
buddha	85000	6	10	26	18	54
thai-elephants	592084	4	40	80	127	247

Table 3.1: *Timings for feature detection on a representative scan of several data sets.*

graph that connects each surface point to its 20 nearest neighbors. Second, we run a standard Dijkstra algorithm to compute the distance from each keypoint to all other surface points, which yields approximate geodesic distances between all keypoints.

3.3.3 Detection of Symmetric Features

To illustrate the distinctiveness of feature matches, we have implemented a simple symmetry browser: The user may select on feature on the object and the application returns all features that have been found on the same object with a matching local neighborhood. Due to the rotational alignment, that we obtain by local ICP as described above, we also have a rigid transformation associated to each matching feature pair. This is not a complete symmetry detection method but more a performance test with respect to symmetry detection. In Section 3.5.1 we will describe a more sophisticated symmetry technique based on slippage features.

3.4 Results

3.4.1 Feature Stability

In real world scenarios we have to deal with noise in the data. We investigate this in the following test. Comparable to [Li and Guskov, 2005], we add uniform noise to the point positions in the normal direction with amplitude 0.5% of the maximum side length of the bounding box and perturb the normals between -0.05 and 0.05. We compare the features found in the noisy data set with the original ones by counting the number of corresponding features. Two features correspond, when their Euclidian distance is less than $\sigma/2$ and their scale differ less than 1.

Figure 3.8 shows the result of this comparison for the well known Stanford bunny and happy Buddha datasets. On the coarser scales one can see that most features can establish a correct correspondence. The peak in the last scale level can be explained by the multi-scale mean shift procedure: All features detected in the coarsest scale shift in the lower ones because the density still increases. If we would compute the a further scale level, these features would appear on their proper scale, but the truncation of the scale level makes all larger scale features occur at the final scale.

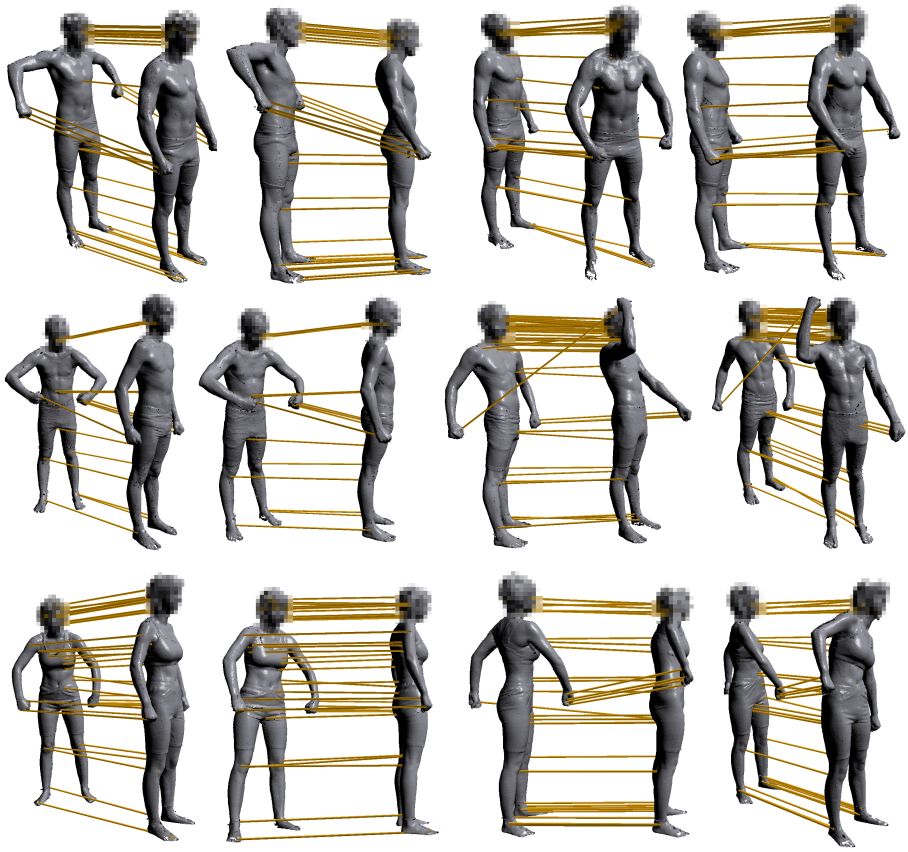
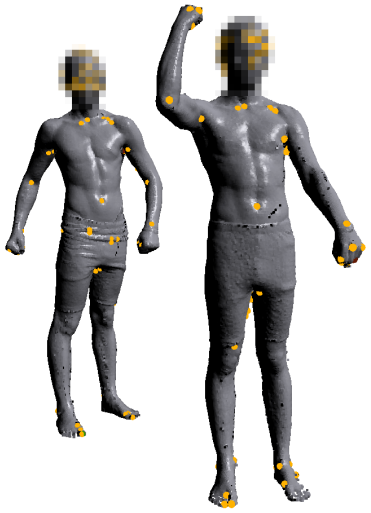
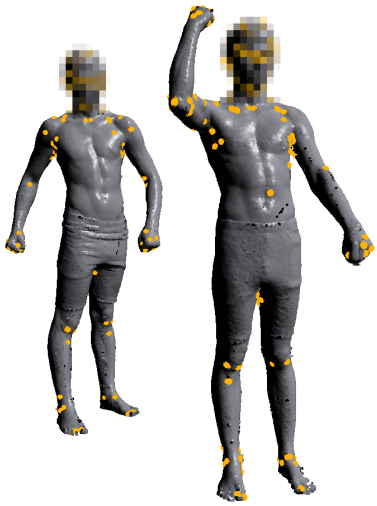


Figure 3.5: *Result of the registration technique with deformable surfaces. In order to protect the identities of the scanned people, we pixelized the face regions.*

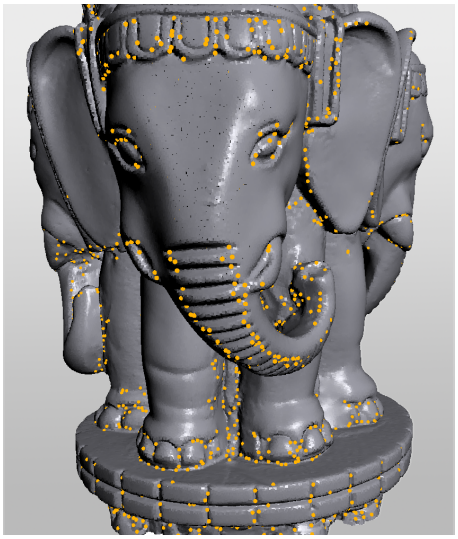


a) DoG features



b) Slippage features

c) DoG features



d) Slippage features

Figure 3.6: *Comparison between SIFT-like geometric features (DoG) (a,c) and slippage features (b,d).*

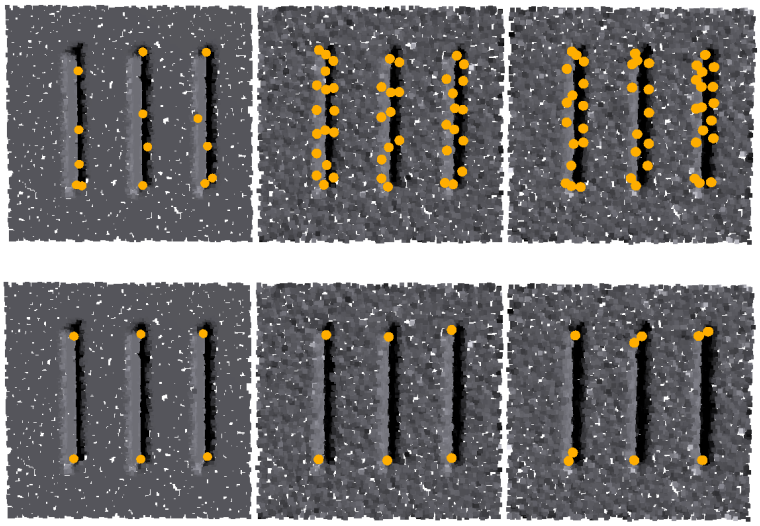
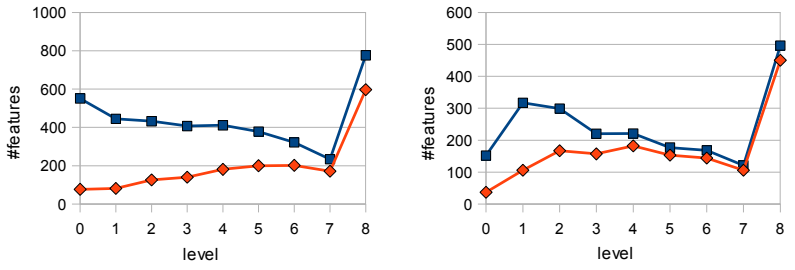


Figure 3.7: *Comparision between SIFT-like geometric features (top) and slippage features (bottom) on a test example. Original example data (left), two noisy copies with noise (middle,right).*



a) Buddha b) Bunny

Figure 3.8: *Feature stability for the datasets happy buddha (a) and bunny (b). All features (blue), features with correct correspondences (red).*

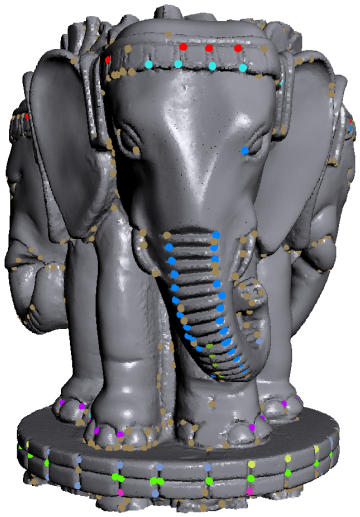


Figure 3.9: *Symmetry browser example: Features which are symmetric to a manually selected feature are shown in the same color (pale brown is unassigned).*

As we use the same technique for the stability analysis, our results are directly comparable to the curves in [Li and Guskov, 2005]. We obtain a similar stability of our generalized features, which shows that the generalization does not harm the stability of the results.

We also compare the feature matching results to those form [Li and Guskov, 2005]. For this, we have reimplemented the technique closely following the description in the paper. Figure 3.7 shows the results of our technique and the SIFT-like approach. Notice how the slippage features appear robustly at the ends of the notch while the other approach is unable to establish stable features, especially under noise (we used a simple thresholding to remove noisy features on both techniques, with thresholds decreasing from left to right). This behavior is expected, as a maximum curvature criterion of the SIFT-like features is not well suited for lines of constant curvature. This effect can frequently be observed in a real data example, such as the well-known Thai Statue data set depicted in Figure 3.6 c),d). Please note that a more stable feature detection is not restricted to the marks on the elephants trunk but we also obtain more and more reliable matches at the ornamental structures. This improved performance is critical for deformable matching (Figure 3.6 a),b)): Using a threshold that yields stable and reliable keypoints, the SIFT-like features are not able to detect keypoints on all extremities of the model. If we decrease the threshold such that correspondences are detected everywhere, the positions of the keypoints become more random and unstable so that identifying corresponding points becomes much more complicated.

3.4.2 Global Registration

We tested our registration pipeline for rigid alignment on the datasets bunny and happy buddha. The most time consuming step is the computation of the slippage features. Timings can be seen in table 3.1 for a single view. For rigid registration we do not need to perform the full feature matching pipeline: We have skipped the ICP-based local alignment and used only the histogram based signatures to establish a set of initial correspondences. False positive matches are handled by the global validation algorithm. The results are shown in Figure 3.4. Please note that we obtain fairly accurate global registration results without an ICP alignment of the parts; this shows that keypoints are retrieved accurately at corresponding positions.

3.4.3 Deformable Registration

We applied our technique to a set of scanned humans. Each individual was scanned in several poses. We established a set of correspondences between two randomly selected poses. The results are shown in Figure 3.5. Unfortunately, the usage agreement of the data set requires not to show faces of the scanned individuals, so that we have to blur out the corresponding parts of the images. Nevertheless, we obtain a similarly consistent registration in the facial area. Notice that every extremity receives at least one correspondence, which is difficult to obtain with feature detectors that look for maximums in surface curvature. These global correspondences could be used as guidance for local, dense deformable registration algorithms such as [Wand et al., 2007; Brown and Rusinkiewicz, 2007].

3.4.4 Computational Costs

The runtime of our keypoint detection technique is comparable to that of the SIFT-like features in [Li and Guskov, 2005], depending on the number of scales and points. Figure 3.1 shows the timings for the individual stages of the algorithm.

3.4.5 Detection of Symmetric Features

We tested our simple symmetry browser scheme with the elefants of the thai statue data set. We selected a few features and detected successfully several symmetric parts. Figure 3.9 shows the results. Notice that even in complicated parts like the toes and the beak symmetries could be found.

3.5 Further Applications

In this section, we describe two matching techniques based on slippage features in order to demonstrate the applicability of slippage features: A symmetry detection method [Berner et al., 2008] and isometric registration technique [Tevs et al.,

2009]. The author of this thesis was only partially involved with the implementations and contributed some conceptual ideas to the projects. Therefore, we only give a small overview of the methods and show their results without going into detail.

3.5.1 Graph-based Symmetry Detection

When a shape contains rigid symmetries, we assume that these symmetries are also present in the slippage features. Ideally, a symmetric part will create identical keypoints and descriptors across multiple instances. Based on this assumption, we can formulate a symmetry detection algorithm using the feature representation: We build a graph of features and use graph matching to find symmetric subgraphs.

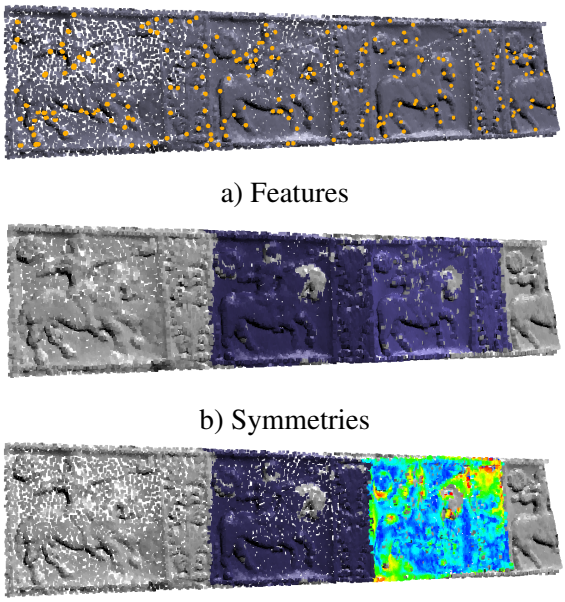
We start by computing a set of slippage features as described above. Then, set up a graph by connecting every feature with its closest k neighbors (typically $k = 20$). With this graph as input, we start a RanSaC-based graph matching algorithm: First, we randomly select a graph edge and find all potentially corresponding edges with similar edge length and feature descriptors. Then, we compute an initial alignment between the selected edge and all corresponding edges (using the reference frames defined by the first feature point of the edge, its normal, and the edge direction) and refine this transformation using ICP in a local neighborhood around the feature points. If the ICP algorithm does not converge or produces a high residual, we consider this pair of edges as not symmetric and reject the second edge. After this pruning step, we have a set of subgraphs (consisting of a single edge). Now, we try to expand all subgraphs with more features that are consistent in every subgraph. This strategy maximizes the number of instances which yield the simplest possible building block. We repeat this process several times to gather multiple symmetries.

So far, we only have a sparse set of feature points that resemble the resulting symmetries. In a final step, we extract the actual point data that is symmetric. We perform region growing on the points, starting with the points closest to feature positions. Region growing stops in two situations, either if the geometry does not match under the symmetry transformation or if the point is already occupied by another instance. For a robust geometry test we utilize moving a least-squares projection technique to smooth the geometry and compare the distance from surface to surface and the surface normals. Additionally, we perform ICP on the current set of points associated to an instance to improve the alignment between instances. Results of the method are shown in Figure 3.10 and 3.11.

The graph matching technique itself is not restricted to point cloud data and can be easily adapted to other types of data (e.g. triangle meshes and images).

3.5.2 Isometric Registration

We assume that surfaces are 2d manifolds that were deformed by an (approximately) isometric transformation. Given two such surfaces, we want to establish



a) Features

b) Symmetries

c) Point-to-plane error (blue = lowest, red = highest)

Figure 3.10: *Result from [Berner et al., 2008]: Scan of a historical artifact of horsemen engraved in stone. The horseman was detected two times. The third horseman was not detected due to deformations.*

a) Features b) Symmetries c) Point-to-plane error

Figure 3.11: *Result from [Berner et al., 2008]: Scan of a handcrafted clay house. Most salient symmetries were detected in this example. Only small windows and reflective symmetries were not recognized.*

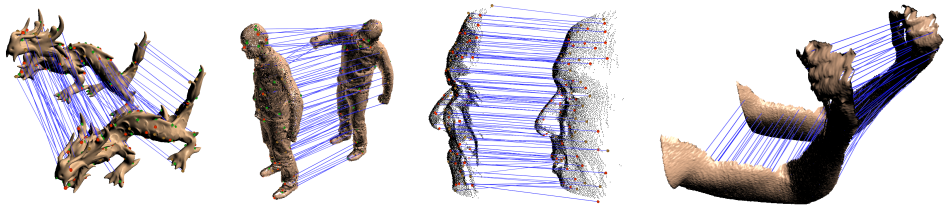


Figure 3.12: *Results with isometric matching [Tevs et al., 2009] of various datasets: a) is the reconstruction of clay figure resembling a dragon that has undergone approximately isometric deformation. b),c), and d) are unstructured point clouds with heavy topological noise and missing data.*

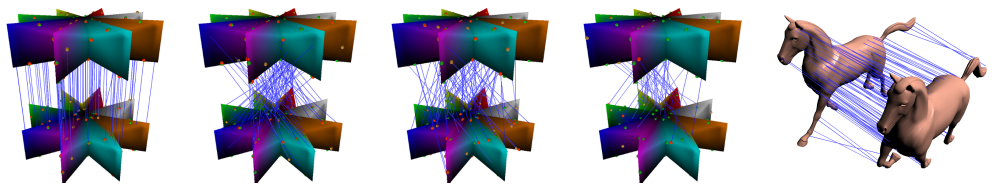


Figure 3.13: *Isometric symmetries using isometric matching [Tevs et al., 2009]. In the synthetic example "star" we obtain identity deformation a) as best solution and a set of symmetric solution b),c),d) (only a few a shown here). In many cases we find the reflective solution as in the horse data set d).*

dense correspondences from one surface to the other. There are several challenges that we want to address here: Scanned surfaces contain noise, holes, occlusions, and do not provide reliable topological information. Depending on the type of scanner and the chosen viewpoint, there might be only a part of the surface that corresponds to the other. Consequently, this type of problem requires a partial matching algorithm that is robust to topological and geometrical noise.

First, we compute slippage features as described above and compute geodetic distances between features (we use a approximation of geodetic distances based on a k-nearest-neighbor topology on the point cloud). The input for the next step are two sets of feature points from surface S_1 and S_2, and two sets of geodetic distances between features (one for each surface). Then, we perform a RanSaC-like randomized sampling algorithm: We start with a random feature from surface S_1, choose a corresponding feature from S_2 based on the feature descriptor and add this correspondence to a set of valid correspondences C. Now we try to successively add random correspondences to C under the condition that at least n geodetic distances to correspondences from C must match in both surfaces ($n = min(|C|, k)$). By requiring only a subset of valid features to agree with a new correspondence, we bypass the problem of topological noise and allow correspondences in the valid set that might be in conflict with some correspondences but have sufficient support from others. Instead of uniform sampling we bias the selection of new correspondences towards correspondences that are more likely to be valid based on the assumption that all correspondences of C are correct. We stop when we cannot add new correspondences to the set and repeat the algorithm multiple times. In each iteration, we keep only the largest set of correspondences C. In Figure 3.12 we show some of the results.

Naturally, the method also returns mirrored solutions since the main verification criterion is based on geodetic distances which are invariant under reflection. Here, we do not apply the standard RanSaC approach that rejects all correspondence sets and keeps only the best. With a slight modification of the method, we also keep a set of different solutions with sufficiently large score as shown in Figure 3.13.

3.6 Summary and Future Work

In this chapter, we presented a method for reliable detection of keypoints based on slippage analysis. Our key concept is to find spherical regions that are locally maximal constrained with respect to the auto-alignment of the encapsulated geometry. A strong benefit of this strategy is that the local neighborhood of a keypoint can usually be robustly registered against the neighborhood of a corresponding keypoint. In many situations, we observed that the proposed method is able to detect more stable keypoints than traditional techniques. Overall, we obtain a conceptually clean framework to detect the very general class of feature correspondences characterized by well-defined translational and rotational geometry matching, avoiding

any further assumptions such as local maxima in surface curvature.

In future work, we would like to investigate the potential use of the other eigenvalues from the slippage analysis. To some extend, we already did this by considering regions with one slippage motion which we will present in the following chapter. However, it might be rewarding to study the combination of all eigenvalues in order to generalize the concept even more.

The ability to compute alignments between single feature points could be utilizes for a large range of techniques. For example, a promising idea would be to combine slippable features with the generalized Hough transform [Ballard, 1981] for object recognition in 3D point clouds.

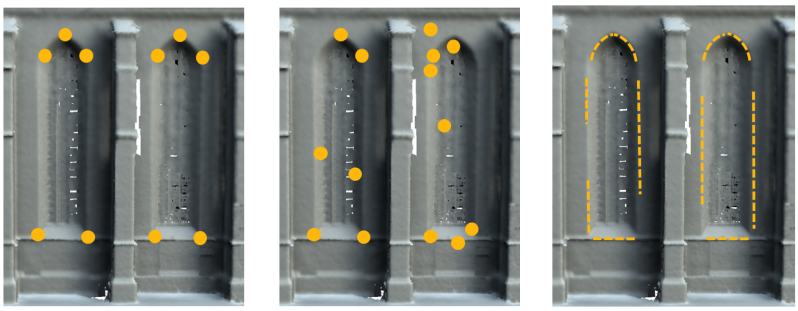
4 Symmetry Detection Using Feature Lines

If you want to find the needle, burn down the haystack.

unkown author

In the previous chapter, we have presented a feature detection technique for keypoints in unstructured point cloud data and described applications to rigid and non-rigid registration techniques and symmetry detection. In the context of global shape registration, keypoint-based methods are an excellent choice and have been used in several ways in geometry processing [Li and Guskov, 2005; Gelfand et al., 2005; Tevs et al., 2009] and in computer vision [Lowe, 2004; Snavely et al., 2006]. However, the keypoint-based approach for symmetry detection [Berner et al., 2008] demonstrates that keypoints might not be the optimal choice for this type of application.

We will illustrate the shortcomings of keypoints with the example of two windows shown in Figure 4.1. In the ideal case we obtain keypoints at the window corners, identically in both windows, that allow us to establish correspondences between the windows. But even in this idealized case we observe that a large area is empty in the feature representation and we need to test the point cloud data directly to verify that both windows are the same. In reality, a keypoint detector does not deliver a perfect result. Some (important) keypoints will be missed while a set of keypoints appear that relate so small surface variations, noise, and holes in the data (illustrated in Figure 4.1 b) which makes symmetry detection much harder in this represetation. The fundamental problem behind keypoints is that they cannot capture the complete surface structure since they are restricted to *points*; linear or planar structures are not represented. If we think of lines as features, such as depicted in Figure 4.1 c), we get a much more intuitive representation. Conceptually, keypoints like corners are implicitly represented by two crossing lines and even planar structures are present in the representation. Othake and coworkers showed, that a surface can be reconstructed from ridge- and valley-lines with normal information [Ohtake et al., 2004]. Furthermore, it is not necessary to extract every



a) b) c)

Figure 4.1: *Illustrative comparison between keypoints and features lines: Ideally, we would like to find the corners of the two windows as shown in a). However, due to noise and holes in the data and subtle differences of the two windows we usually miss some keypoints while a large number of noise keypoints appear b), that do not belong to a real geometric feature. In contrast to kepoints we can also considering lines as features c). Among other advantages, line features benefit from a high probability to be found due to their large spatial extend.*

feature line completely without interruptions to recognize similarity between the two windows because lines can also be extrapolated locally. This property immediately yields a certain robustness against noise and missing data since we only need to find parts of a feature line. Consequently, we propose to use *feature lines* instead of keypoints.

In this chapter, we present a novel symmetry detection method for rigid symmetries based on line features. As input, we expect an unstructured point cloud that resembles a surface and our goal is decompose this surface into parts and a set of rigid transformations that map these parts onto the surface. Intuitively, we want to obtain a set of building blocks, each with a list of transformations, that reassemble the original geometry up to a certain tolerance. The most successful techniques for detecting such types of symmetry are based on transformation voting, where transformations between candidate pairs of potentially corresponding points are inserted into a Hough-Transform space to vote for dominant transformations [Mitra et al., 2006b; Loy and Eklundh, 2006; Podolak et al., 2006; Mitra et al., 2007; Pauly et al., 2008]. This approach is very elegant and gives good results in many cases. The main limitation is however that all transformations end up in the same voting space with spatial correlation information lost so that the problem of isolating the relevant symmetries becomes harder when the number of different symmetries grows [Pauly et al., 2008]. As a consequence, these techniques have so far only been applicable for coarse, large scale symmetries [Mitra et al., 2006b; Podolak et al., 2006], in special cases were the model can be decomposed hierarchically [Mitra et al., 2006b], or if the symmetries form regular patterns in transformation space [Pauly et al., 2008].

The goal of this work is to detect a large number of symmetries without such

Figure 4.2: *Line features for the "old town hall" data set. Without knowing the input object, we can easily see that it is a building of some sort using only line features as source of information. Furthermore, we also can guess the buildings structure and symmetry properties. We use this representation to detect rigid symmetries efficiently.*

prerequisites. The key observation for an improvement in recognition capabilities is that we can expect some spatial coherence: The instantiated part, which is found several times in the model under a symmetry transformation, will typically itself form a localized, coherent spatial object: Nearby points are likely to show the same symmetry pattern. This information is lost in transformation space.

In order to exploit this information, we choose a feature-based approach: First, we extract line features from the input surface and form a spatial neighborhood graph of such features. Next, we examine the graph for recurring patterns using a randomized matching algorithm. Thereby, we directly utilize spatial coherence of symmetric parts. Finally, the results from matching local clusters of features are transferred and validated on the original geometry. The core idea is closely related to the graph-based matching technique [Berner et al., 2008] described in the previous section but differs in the matching strategy as well as in the feature type. The new strategy yields both a substantial improvement in recognition rates as well as a significant improvement in running times. As an example application for our technique, we apply the algorithm to symmetry-based automatic reconstructing of scanning data [Gal et al., 2007; Pauly et al., 2008].

We make two main contributions: First, we propose a novel symmetry detection algorithm that can find rigid symmetries in general configurations. The algorithm works efficiently on large data sets, up to two orders of magnitude more complex than previous state-of-the art results. Second, we develop a framework for rapid geometry matching based on sparse sets of feature lines.

This chapter is based on the paper [Bokeloh et al., 2009]. The author of this thesis contributed the main conceptual ideas and most of the implementation. How-

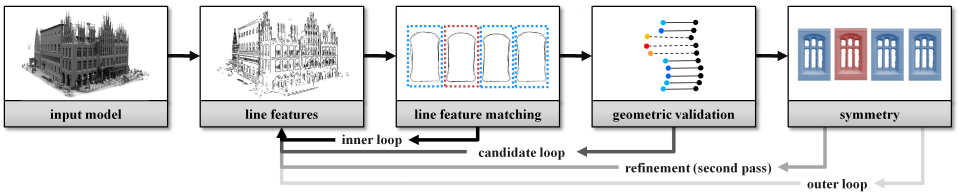


Figure 4.3: *Overview: We extract line features from the input model. Inside the inner loop, we search for symmetric constellations of line features and pass the best candidate to the validation stage, where we verify it taking all data points into account. We repeat this process several times inside the candidate loop and return the best candidate (one symmetric part with its instances). This set is then refined, in order to find more instances. An outer loop repeats the process to find several symmetries.*

ever, the implementation of Section 4.4 (Geometric Validation) and Section 4.6 (Reconstruction) was done by Alexander Berner and should not be seen as part of this thesis. In order to provide the reader an accurate and complete picture of the method, we will describe the whole approach.

4.1 Algorithm Overview

Our algorithm consists of three main stages (see Figure 4.3): *Feature detection* (Section 4.2), *line-feature matching* (Section 4.3) and *geometric validation* (Section 4.4). The last two steps are used iteratively in an outer loop (Section 4.5). Finally, the symmetry information is then used for scan reconstruction and other applications (Section 4.6).

The first stage extracts line features and identifies suitable pairs of these features to define local coordinate systems that we call *"bases"*. The second stage then tries to find subsets of such bases so that line features in their local neighborhood match. In order to verify the matching, the algorithm uses an *"iterative-closest-line (ICL)"* algorithm that aligns and compares sets of line features. The rationale behind this design is that a comparison of a sparse set of line features is substantially more efficient than comparing actual geometry, while still achieving a very high matching accuracy. Probably even more important, matching of line features is more resistant to holes and outliers in the data than conventional geometry alignment techniques such as ICP [Chen and Medioni, 1992; Besl and Mckay, 1992] (see Figure 4.6).

In the next stage of the algorithm, geometric validation, we transfer our findings to actual geometry, i.e. the input point cloud. As the line feature representation is sparse, this step is necessary in order to correctly assign geometric regions to found instances. It computes disjoint instances of geometry that are replicated by the symmetry transformations. Line feature matching and geometric validation are repeated in a "candidate (RanSaC) loop", outputting the best match only, to

avoid spurious matching results. Once we know the actual instances of symmetric geometry, we refine our results in a second matching pass to increase the number of recognized instances: We detect additional bases belonging to the instances by checking for overlap with the points computed in the geometric validation stage. In addition, we also try to extrapolate known transformations to discover possibly undetected parts in a transformation prediction step. Afterwards, geometric validation is performed again. This yields a single symmetric part with all its final, typically larger, set of instances. Finally, the outer loop of the algorithm iterates this whole procedure several times in order to find several different classes of symmetry in the model.

4.2 Feature Extraction

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

This problem can be addressed by slippage analysis of the local geometry [Gelfand and Guibas, 2004]. Slippage analysis determines whether the problem of aligning a piece of geometry with itself is well posed: It sets up an ICP objective function (the sum of all point-to-plane surface distances of surface points with themselves), with 6 degrees of freedom (3 translational and 3 rotational). The eigenvalues of the Hessian of the objective function determine whether the geometry is *slippable*: Any surface is necessarily constrained in at least 3 degrees of freedom when being matched to itself. The eigenvectors of the Hessian with eigenvalues (numerically close to) zero, here called *slippage vectors*, describe the associated degrees of freedom. The slippage vectors may mix translational and rotational degrees of freedom (see [Gelfand and Guibas, 2004] for details).

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

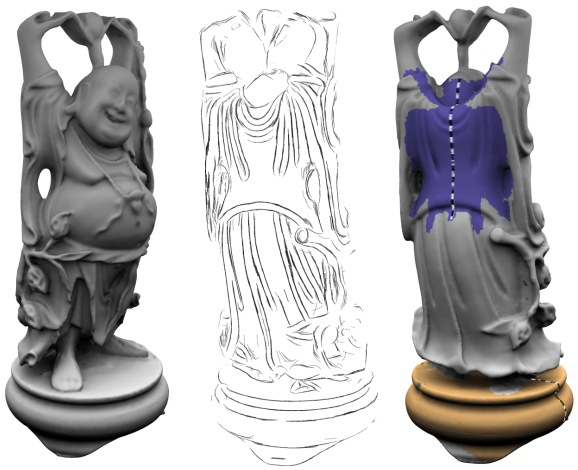


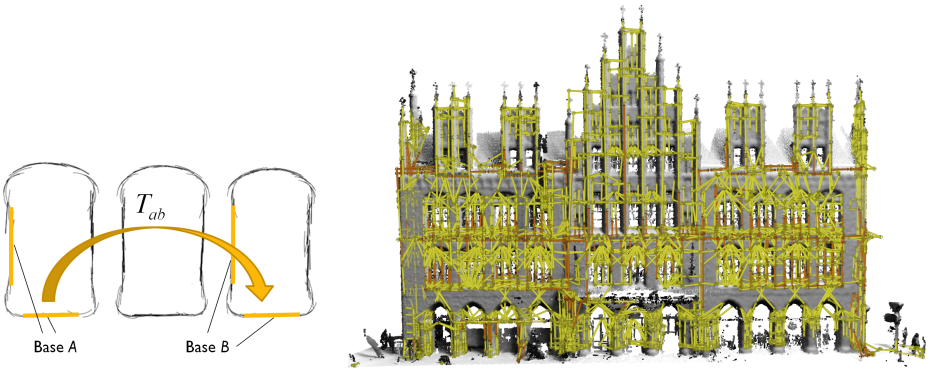
Figure 4.4: *Line features for the Happy Buddha model (middle) and two detected symmetries (right).*

Here, we only consider the case of pairs of feature regions that are still slippable in one dimension, with a non-zero translational component. This corresponds to regions with arbitrary cross section extruded along (locally) straight or circular lines. As such regions will have a one-dimensional symmetry direction, we refer to them as *line features*. In practice, 1D slippable feature lines contain most of the salient information and thus provide a both compact and efficient representation. They work particularly well for man-made objects such as architectural models (Figure 4.2), but we also obtain descriptive results for many natural objects such as faces or sculptures (Figure 4.4). Our goal is now to extract such 1D-slippable regions from the input geometry. As being slippable in one direction, these regions will have a linear structure.

4.2.1 MLS Line Features

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

As input we expect a point cloud $\tilde{\mathcal{S}} = \{\mathbf{x}_1 ... \mathbf{x}_n\}$ sampled from a smooth manifold $\mathcal{S} \subseteq \mathbb{R}^3$ where each point $\mathbf{x}_i$ is equipped with a normal vector $\mathbf{n}_i$. We now define the projection operator: In order to project a point $\mathbf{p}$, we first compute a local slippage analysis of $\mathbf{p}$ (we use a Gaussian window function centered at $\mathbf{p}$ with standard deviation ϵ_{feat}). Slippage analysis gives us a set of unit-norm eigenvectors $\mathbf{v}_1, ..., \mathbf{v}_6$ that represents slippable motions and the corresponding eigenvalues $\lambda_1 \leq \ldots \leq \lambda_6$. We are only interested in the first eigenvector $\mathbf{v}_1$ that represents the most slippable rigid motion. From $\mathbf{v}_1$ we extract the translation part $\mathbf{s}$, which



a) Example bases b) Bases of 'Old Town Hall'

Figure 4.5: *A base consists of a pair of feature lines with different slippable motions. We can easily fix a coordinate frame to each base a) and thus compute transformations between bases. For the front side of the 'Old Town Hall' of Hannover a visualization of all bases is shown in b). A base is rendered as two arrows pointing from the base implicit corner (the point with minimal distance to both feature lines) to the feature lines defining the base. For the special case of two straight feature lines we show the base in orange color.*

becomes our estimate of the tangential direction of the line. We now find a local maximum of the mean curvature along the direction $\mathbf{s} \times \mathbf{n}$. We use a simple gradient descent algorithm that stops were the gradient vanishes. To compute the mean curvature of a given surface patch we fit a local quadratic patch in a least-squares sense, again using the same Gaussian window function, and compute the eigenvalues of the quadratic component. We also recompute the slippage analysis at the final feature point and store the first eigenvector.

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

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

4.2.2 Building a Feature Graph and Bases

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

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

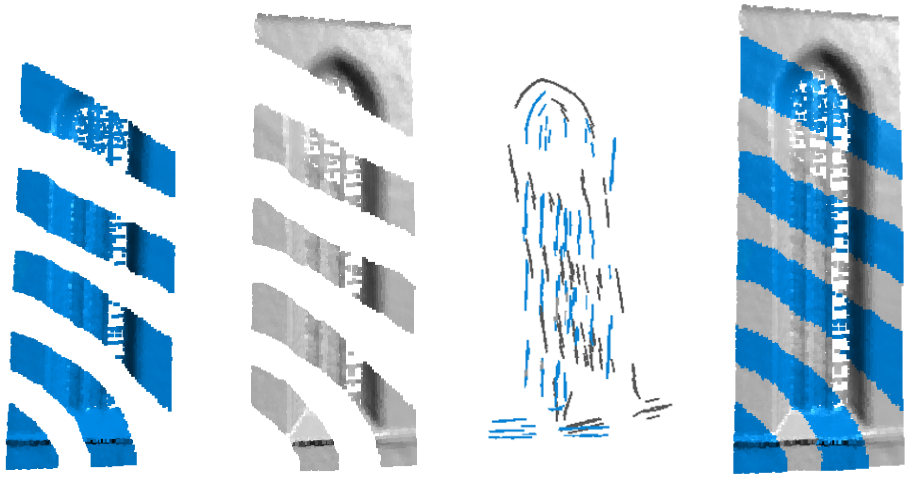
From these line clusters, we now form *bases*, which represent local coordinate frames that will be used as matching candidates in later steps of the algorithm. Bases can be interpreted as corners implicitly defined by two nearby lines. Again, we walk along the graph and find all edges that connect between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

4.3 Line-Feature Matching

Having obtained a set of feature lines, our symmetry detection algorithm tries to find local constellations of these feature lines in different parts of the model that are similar. We use two steps: First, we identify a set of base matching candidates to estimate initial transformations. Second, we examine the line features in the neighborhood in order to verify the match. These two steps are iterated in a RanSaC like matching loop in order to find the best matches (this is the inner loop in Figure 4.3).

4.3.1 Base Matching

In order to find matching bases, we pick a random base out of the previously extracted set. Please note that large symmetries with many instances are automatically preferred in the random pick as they contain more bases. The base consists of two line clusters with different direction of slippability, therefore defining a local coordinate frame. Next, we find all other bases that could be potential matches,



a) Part 1 b) Part 2 c) Initial pose d) Aligned result

Figure 4.6: *ICL example: Aligning two disjoint subsets of a 3D scan. We compute line features for both parts and run ICL on the example configuration shown in c). The registered result has a maximum alignment error of 0.6% of the maximum bounding box side length.*

regarding only local information about the bases. For this, we build a simple feature descriptor: For each base, we store the curvature of the two lines involved and the average mean curvature of the surface points of the underlying geometry. We use a matching threshold for these values of $0.1/\epsilon_{feat}$. In case both lines are straight lines with no curvature, we also include the angle at which the two lines meet in order make the criterion more discriminative. For each of these candidates with matching descriptors, we now compare the lines in its neighborhood by an alignment algorithm. Before going into details of the procedure, we first discuss the alignment algorithm:

4.3.2 Iterative Closest Lines (ICL)

Our iterative closest line (ICL) algorithm is a straightforward generalization of the point-to-plane ICP algorithm [Rusinkiewicz and Levoy, 2001] to aligning line segments. The idea of sampling well constrained regions is also similar in spirit to the adaptive ICP technique in [Gelfand et al., 2003]. We assume that we are given a point-sampled representation of curved lines: We have a set $\{\mathbf{x}_i\}$ of sample points on the lines, each equipped with a unit tangent $\mathbf{t}(\mathbf{x}_i)$, a unit normal $\mathbf{n}(\mathbf{x}_i)$ that describes the normal of the surface the line lives in. By construction, these two are orthogonal. From this, we also compute the normal $\mathbf{b}(\mathbf{x}_i) = \mathbf{t}(\mathbf{x}_i) \times \mathbf{n}(\mathbf{x}_i)$ to the feature curve in its tangent plane. Our goal is now to align two different sets of curved lines $\{\mathbf{x}_i^{(0)}\}, i = 1..n$ and $\{\mathbf{x}_j^{(1)}\}, j = 1..m$. Similar to the ICP algorithm,

we first compute the closest point from the set $\{\mathbf{x}_j^{(1)}\}$ to each point from $\{\mathbf{x}_i^{(0)}\}$. We denote the index of the closest neighbor by $N(i)$. If the closest neighbor is too far away, we simply discard it in the current iteration. Given the correspondences, we minimize the following energy function:

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{n} \omega\left(dist(\mathbf{x}_i^{(0)}, \mathbf{x}_{N(i)}^{(1)})\right) \cdot \left[dist(\mathbf{x}_i^{(0)}, \mathbf{x}_{N(i)}^{(1)})\right]^2 \tag{4.1}$$

with

$$dist(\mathbf{x},\mathbf{y}) = \left\| \begin{pmatrix} | & | & 0 \\ \mathbf{n}(\mathbf{x}_i) & \mathbf{b}(\mathbf{x}_i) & 0 \\ | & | & 0 \end{pmatrix} (\mathbf{x} - (\mathbf{R}\mathbf{y} + \mathbf{t})) \right\| \tag{4.2}$$

where ω is a one-dimensional Gaussian window function, $\mathbf{R}$ is an unknown orthogonal matrix and $\mathbf{t}$ an unknown translation. The first matrix in Equation 4.2 is a projection matrix that removes the tangential component from the distance vector connecting two line segments, thus only measuring the point-to-line distance. For the optimization, we use the standard approach of locally linearizing the manifold of orthogonal matrices and solving iteratively [Mitra et al., 2004].

In order to improve accuracy and robustness we also discard correspondences with different tangent vectors with respect to the current rotation. Additionally, we also take line feature properties into account like line curvature, surface curvature, and surface normal to filter out invalid correspondences.

4.3.3 RanSaC Search

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

Each iteration starts with computing a random base B_0, and a set of potentially matching bases $B_1 ... B_n$ for which the descriptors match. Next, we compute a pairwise matching score comparing the neighborhood of B_0 to all other bases B_i separately. Matching the neighborhood lines of two bases proceeds in several steps: First, we align only the lines that form the bases B_0 and B_i themselves, which can be computed analytically. At this point, the feature line representation has the advantage that matching pairs of lines already gives a stable initial guess, unlike feature point matching [Huang et al., 2006b; Berner et al., 2008]. Then ICL is used to refine the match. After aligning the lines of the bases B_0 and B_i, we walk on the graph of adjacent line segments in a region growing algorithm. We start by putting all line segments adjacent to those in B_0 on a priority queue sorted

by distance. We then subsequently retrieve the next closest line segment from the queue and compute the weights of the point-to-line distances (Equation (4.1)) for the transformed lines. If the resulting weight ω (which now serves as *matching score* for this segment) is smaller than a fixed threshold (typically: 0.02), we discard the segment. Otherwise, we add the weight to the overall matching score of B_i and put its neighbors on the queue. The algorithm stops if the queue is empty or a maximum distance is reached (typically: 10% of the model size), to limit the costs.

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

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

$$score = matching\_score_{av} \cdot \#instances^2 \tag{4.3}$$

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

4.4 Geometric Validation

So far, we have done all computations on the line features of the surface $\mathcal{S}$ only. This usually gives a good indication of actually corresponding geometry but we

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

4.4.1 Basic Region Growing

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

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

4.4.2 Grid Based Growing

The basic region growing algorithm with its test of every point is too expensive for huge models. We improve the performance by looking at surface pieces at once rather than isolated points. We impose a regular grid onto the urshape and treat all

points within one grid cell simultaneously. In particular, the decision to include or not include a piece of geometry is now made per voxel cell, rather than per point. When the basic algorithm compares a piece of geometry, we set the radius r_{comp} to $2\times$ the grid cell diagonals. Please note that the voxel grid is imposed on the urshape only and not on the whole model; the geometry in the voxel is transformed to the instances for comparison. This avoids aliasing problems in the that would occur if we were comparing pairs of voxels.

4.4.3 Handling Holes

One application of symmetry detection in 3D scanner data is filling up acquisition holes and equalizing sampling density. Therefore, we need to be robust to acquisition holes. Our solution is to check the number of points within the spheres $S^{(i)}_{comp}$. If too few points are found (typically: less than 6), no reliable plane fit is possible and we treat the voxel as a "hole". We use a relaxed threshold for outlier mismatches due to holes (up to 30% in terrestrial scanner datasets), allowing for matching partial data more robustly.

4.5 Candidate Loop and Outer Loop

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

4.5.1 Candidate Loop

We can make this basic strategy more reliable by again employing a RanSaC-like randomized sampling approach: Instead of removing and thus finalizing symmetric geometry after finding one new set of matches, we first compute a larger number of matches (typically 5 matches), including geometric validation (the refinement step described in the preceding paragraph is actually done *after* the candidate loop, on its result, to safe some computation time; see Figure 4.3). From the candidate set, we pick only the best match. The score for this decision is computed as number of (voxel-quantized) sample points in the urshape multiplied by the number of instances squared (similar to Equation (4.3), again preferring many instances over few instances with many points. The outer RanSaC loop stops when a fixed number of outer loop iterations has passed. We do not accept symmetries covering fewer than a minimum number of data points (less than 1,000) to avoid spurious matches. This prevents outputting "garbage" solutions after all detectable symmetries have been found. An interesting side effect of this strategy is that the algorithm will

output the "most important" symmetries first, which are symmetries with many instances covering a lot of area.

4.5.2 Refinement

A drawback of the algorithm as presented so far is that it can only detect a class of instances that all contain the same basis (in the sense of a pair of lines with the same curvature, placed roughly at the same spatial location). In order to improve the recognition performance for difficult data sets, we now include adjacent bases into the search (this is the "second pass" in Figure 4.3): After geometric validation, we know the concrete area covered by the instances. Therefore, we can check which other bases are also covered by the geometry and try to match them as well. We compute all bases covered by an instance and project them back into the urshape. In the urshape, we cluster similar bases, i.e., consisting of approximately the same lines. For each cluster, we maintain a counter to vote for bases that are contained in many symmetric instances. Having a weighted list of other bases contained in the same instance, we now execute the line matching algorithm of Section 4.3 again, starting at these alternative basis. Instead of region growing, we directly fix the area to be checked by the line segments that fall onto the urshape when being back-projected. In order to limit the computational costs, we do not check all alternative bases (which can be thousands), but rather sample a small number of bases (typically 5) randomly, with probability proportional to the number of votes for each basis. After this step, we execute region growing again, now including all newly found instances.

We can improve the recognition rates further using a prediction heuristic, inspired by [Pauly et al., 2008]: We consider the relative transformations between instances that are spatially close and recursively check whether we can find a matching line constellation if we apply this transformation again to an instance at the boundary of the detected area (a candidate transformation at a boundary will transform a base such that it does not come close to an already known base).

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

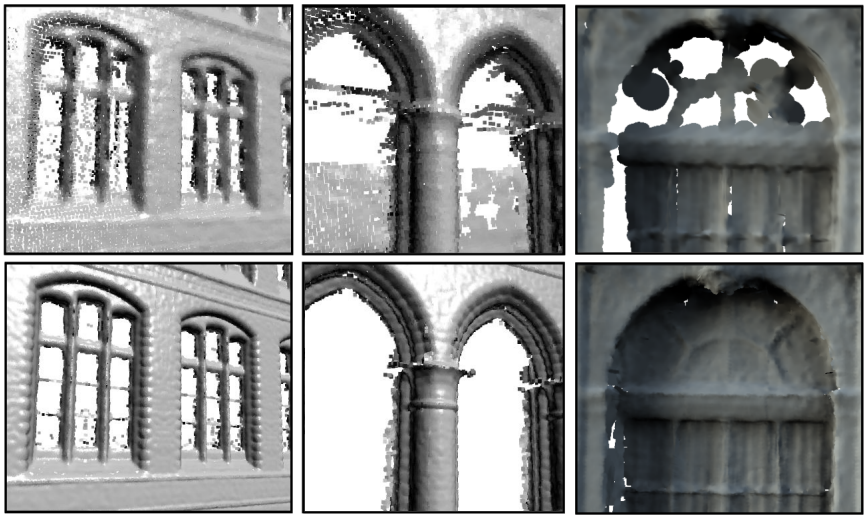


Figure 4.7: *Reconstruction examples for the data sets old town hall and Zwinger (rightmost).*

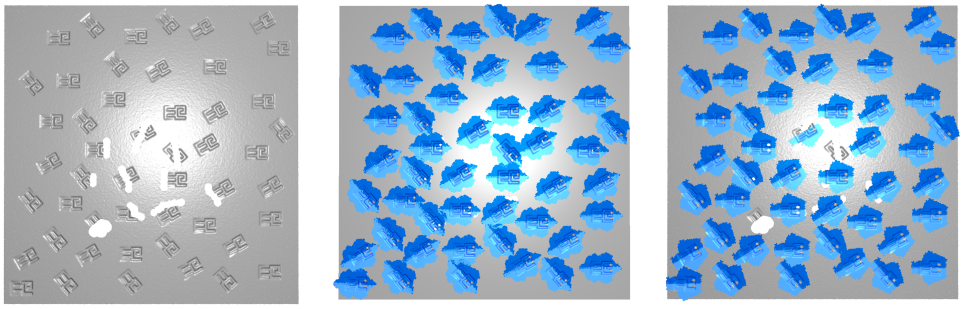
4.6 Applications

As an application of our symmetry detection algorithm, we consider an automatic reconstruction-by-symmetry algorithm, in the spirit of [Thrun and Wegbreit, 2005; Pauly et al., 2005; Gal et al., 2007; Pauly et al., 2008]. Many real-world range scans suffer from strongly irregular sampling, noise and acquisition holes. We will automatically detect similar parts and compute a high density average to improve the sampling quality in regions where such redundant information is available.

Our reconstruction technique first computes an improved urshape and then transforms the improved version back into the instances, replacing the distorted geometry. We start the urshape improvement by copying all points of all instances into the urshape, taking all points into account that fall into one of the urshape voxel cell under the instancing transformation. From this set, we remove outlier points. Outliers are points that show up in less than 30% of the instances. For this co-occurance test, we use a small sphere around the sample point, according to the scanner sample spacing, and check whether the instance provides such a supporting point within that radius. The remaining non-outlier points are then smoothed using a quadratic MLS approximation.

4.7 Implementation and Results

We have implemented the proposed symmetry detection and reconstruction system in C++. Timings have been obtained on a standard PC with Core2 Duo at 2.4Ghz, using a single threaded implementation. In order to visualize the results, we assign a unique color to each symmetry. If two instances of the same type share a border,



a) Noisy data b) Result (no noise) c) Result (with noise)

Figure 4.8: *Test with synthetic data: We engraved the Eurographics logo (EG) into a plate, cut out some holes and applied our recognition pipeline. In the case without noise we obtain a perfect result including the reflective symmetry. Please note that only a small non-reflective part is left out that distinguishes the G from its reflected counterpart. If we add Gaussian noise to the data we miss 5 instances with large holes but still retrieve all remaining instances including a set of instances with smaller holes c) .*

we indicate this by drawing a black/white colored line in between. In the following, we discuss the results for several test data sets.

4.7.1 Synthetic Example

As a synthetic example for completely irregular patterns of symmetry, we have engraved the Eurographics logo into a flat plane, rotated and shifted randomly. For some of the instances, we have cut out holes of different size. Our algorithm is able to detect all of the logos automatically (Figure 4.8); it discovers the reflective symmetry of the logo, leaving non-symmetric parts in the letter G out, and detects all instances of these in the scene. If we add Gaussian noise with a standard deviation of 5% of the maximum heightfield height, some of the instances with big holes are not retrieved anymore but we still obtain good results for most of the symmetric parts.

4.7.2 Scanner Data

We also apply the proposed algorithm to raw 3D scanner data. Our first example is a scan of a museum (Figure 4.9). We are able to recognize most of the apparent symmetries, including most of the possible instances (such as the windows in the front). In particular, we detect instances as belonging to the same class that are not part of a simple regular pattern (see for example the jutty in the middle of the building). A remaining limitation of our approach is the fixed allocation of geometry to a single instance. Therefore, some choices of forming instance preclude detecting some other symmetries (for example: The small window above the jutty,

which is not combined with a larger window below, unlike most other of these small windows). This effect is responsible for most of the missed out symmetries in this example. Please note that we do not obtain any false positives. This shows that the multi-stage filtering strategy of our algorithm is successful in that respect.

Our second test case is a scan of the 'Old Town Hall' in Hannover (Figure 4.10). With nearly 8 million points this is the largest data set we tested. In order to bound the computation time, we use a fixed number of 25 outer loop RanSaC iterations, which yields 19 successfully detected (i.e., large enough) instance sets. In this case, we detect many symmetries with a large number of instances. However, we also obtain some symmetries covering a large area with a small number of instances (for example the facade to the left). These regions mostly consist of small windows with few horizontal lines which makes a detection of the small scale instancing structure harder, so that the initial set (before refinement) is not likely to contain many instances. Therefore, the large symmetry often wins the outer loop RanSaC decision by its area. Again, this problem could be resolved by relaxing the restriction of simple, non nested and non-overlapping instances, which we leave for future work.

The 'New Town Hall' in Hannover (Figure 4.11) is another very large test case for our method. In total, we detect 29 different symmetries including two symmetries with relatively small instance size. For example, the small windows colored in pale red in Figure 4.11 b) have a width of approximately 1 % relative to the bounding box diagonal of the data set. Nevertheless, we successfully detect all 18 instances (9 windows plus reflectional symmetry) without any knowledge about the nested structure. In 4.11 b) we can further observe that the method recognizes the statues as different and does not include them into the symmetries (in the upper part there are three architrave blocks but only two status; in the lower part there are 4 statues that differ). The window segments in 4.11 c) are of comparable size to the red windows of 4.11 b) but with slighty lower sampling density. The method extract 12 out of 16 instances in both towers. The symmetry with the most instances is a reflectively symmetric column of windows without a balkony (bright brown) and was recognized 48 times in the whole building. A problem that arises from the region growing technique is that we miss lots of symmetries due to our decomposition strategy. Consider for example the global reflective symmetry of the 'New Town Hall': In the feature matching stage we find this transformation frequently but then reject the global symmetry and prefer instances that appear multiple times. Since we remove all line features belonging to an instance, we also remove potential candidate transformations. A solution of this problem would be to divide the process into two parts: Symmetry detection, where we build a map of all symmetries for every data point and decomposition using the complete symmetry information.

The last example is the Zwinger data set also used in [Berner et al., 2008]. In comparison to Berner et al.'s approach, we obtain significantly better recognition results (detecting all windows including the reflective symmetry of each window instead of only two instances without reflective symmetry). In Figure 4.13 a com-

Figure 4.9: *Result 'Willhelm-Busch Museum' (courtesy of the Institute for Cartography and Geoinformatics Hannover).*

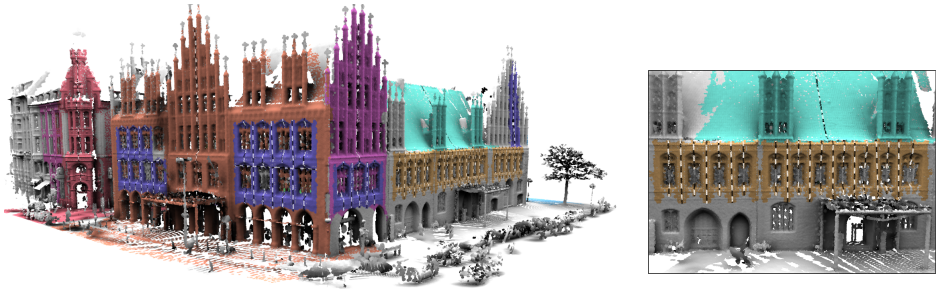


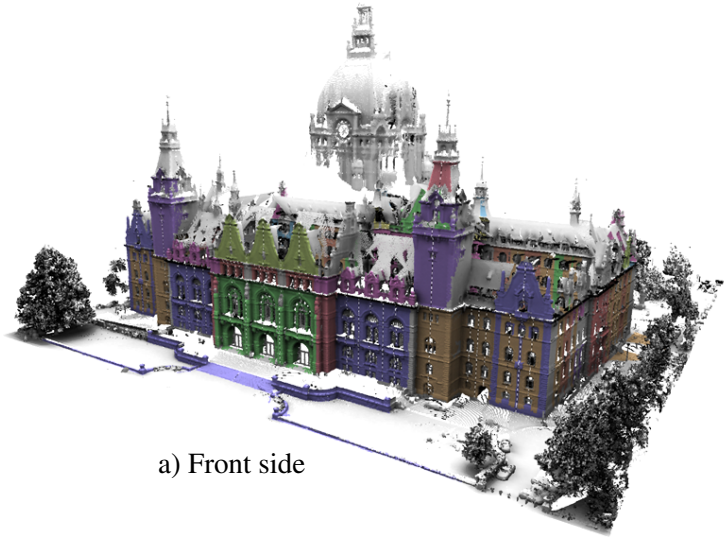
Figure 4.10: *'Old Town Hall' Hannover (courtesy of the Institute for Cartography and Geoinformatics Hannover).*

parison between line features a) and keypoints b) is shown for visual comparison.

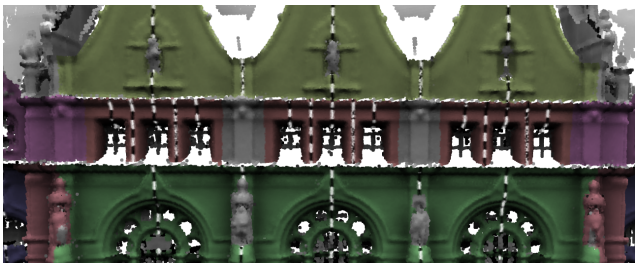
To examine the limits of our algorithm, we have applied it to the Happy Buddha data set. This yields only two small reflective symmetries (Figure 4.4). Line detection and matching works in principle for such data sets, but the main problem in this case is that the data set does not contain larger rigidly symmetric parts.

4.7.3 Reconstruction

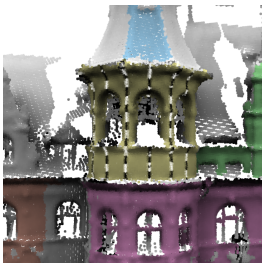
We apply our reconstruction approach to the front of the old town hall and to the Zwinger data set (Figure 4.7). With our symmetry reconstruction technique we obtain a significantly improved quality. Surprisingly, the quality is better than one would expect from just averaging a small number of instances (just leading to square-root error decay). The reason for this is that we do not only average out noise, but also increase the sampling density; in these examples, an insufficient sampling density lead to more geometric uncertainty than the noise in the distance measurements. In addition, we can also fill holes such as the window frames, which are typically acquired from at one side only.



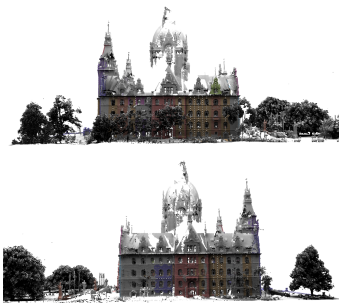
a) Front side



b) Zoom (front side)



c) Zoom (back side)



d) Side views

e) Back side

Figure 4.11: *Symmetries found in the dataset 'New Town Hall' (courtesy of the Institute for Cartography and Geoinformatics Hannover).*

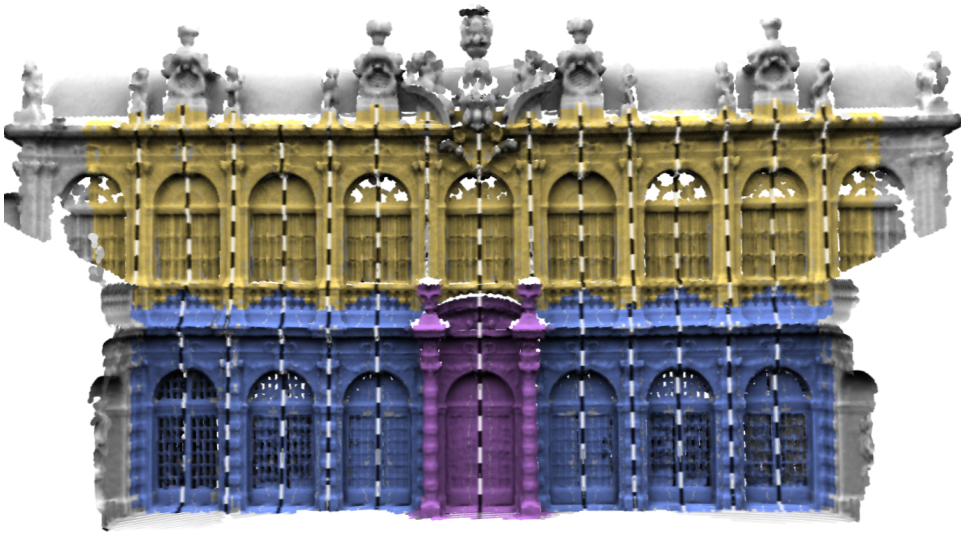
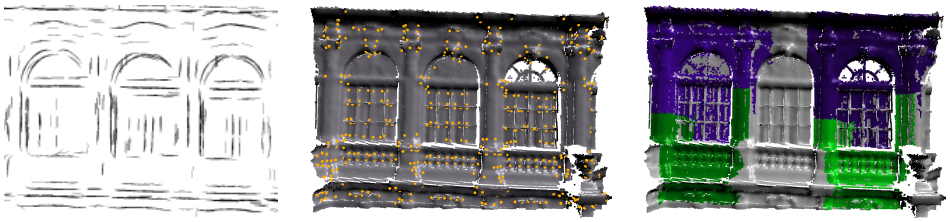


Figure 4.12: *Detected symmetries in a 3D scan 'Dresden Zwinger' (data set courtesy of M. Wacker).*



a) Feature lines b) Keypoints c) Results of keypoint-based method

Figure 4.13: *Comparison between feature lines and keypoints. The keypoint-based method [Berner et al., 2008] detects only two out of three windows, fractioned into two parts.*

4.7.4 Performance

Statistics and timings for the example scenes are shown in Table 4.1. Our largest data set, the old town hall, took about 50 minutes to process, with a little less than half of the time spend in feature detection. In comparison to [Pauly et al., 2008], we are able to handle scenes that are substantially larger than the data sets examined in their paper. In terms of running time, we have comparable requirements, taking the size of the model into account.

4.7.5 Parameters

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

4.8 Summary and Future Work

We have presented a new algorithm for symmetry detection. The main idea is to look for symmetric constellations of feature lines on 3D surfaces in order to find similar parts. In comparison to previous transformation voting algorithms, we avoid the problem of cluttering the transformation space and therefore get a

	# Points	Feature Detection	Clustering + Graph	Symmetry Detection
Plate	880.673	25s	1.1s	1m 59s
Plate (noise)	880.673	1m 12s	3.3s	3m 14s
Old Town Hall	7.735.576	23m 12s	25.8s	32m 40s
New Town Hall	4.598.444	————	81m (total)	————
Museum	2.200.652	4m 44s	5.2s	13m 36s
Zwinger	278.512	32s	2.3s	3m 25s

Table 4.1: *Statistics of the datasets.*

good recognition performance without additional assumptions on the structure of the symmetries. In comparison to previous attempts of using feature points for symmetry detection, feature lines yield a significant improvement in recognition results. As a side effect, the reduction to feature lines reduces the amount of data to be examined substantially, allowing for handling substantially larger models than previous algorithms.

A limitation of our algorithm is that the symmetry decomposition only works if instances are all exactly similar. A direction for future work would be to build morphable statistical models of symmetric parts to better represent subtle variations beyond the accuracy of the region growing algorithm. In addition, one could also examine more general strategies in matching graphs of surface features, beyond rigid mappings.

Another limitation is the strict decomposition strategy that does not allow nested or overlapping symmetries. According to our experience with the different stages of the algorithm, we know that the amount of correct symmetry transformations from the line matching stage is far higher than the actual output and reveals much more symmetry information. In future work, one could develop a method that uses all available symmetry information retrieved by line matching in order to improve the recognition performance. Furthermore, we want to generalize our decomposition strategy to overlapping and nested symmetries and thus capture the objects'structure in a better way.

Part II

Symmetry for Model Synthesis

In Part I of this thesis we have presented feature-based methods for correspondence estimation and symmetry detection. The ability to retrieve this structural information of shapes allows us to understand shapes at a very low level and leads to new way to represent and alter shapes. For example, we can use the presented symmetry decomposition technique to propagate changes of surface attributes to all symmetric instances as shown in Chapter 4.

Now, we want to take a closer look at symmetry structures of 3D shapes and the information that they capture. A shape can contain all sorts of recurring elements, regularities, and global symmetries. Surprisingly, the least useful kind of symmetries in our context are global symmetries while partial symmetries can reveal potential shape modification operations. Especially boundaries of a partial symmetry (that is where the geometry starts to differ) are most interesting in the our context: At this border we might be able to replace geometry with its non-symmetric counterpart. This is a key insight that we gained during this thesis and that we will describe in the following chapters.

In Chapter 5 we present an approach to extract a shape grammar fully automatic from a shape by analyzing its partial symmetries. Then, we will present an interactive deformation technique in Chapter 6 that uses regular patterns in order to preserve visually important structures during deformation.

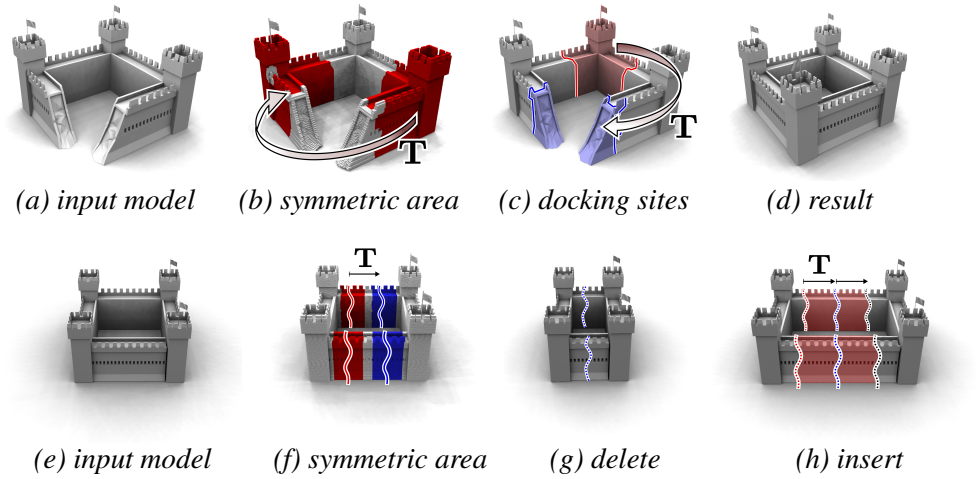
5 Inverse Procedural Modeling

Building one space station for everyone was and is insane: we should have built a dozen.

Larry Niven

Man-made shapes are heavily structured objects and follow a variety of design laws considering esthetical, statical, and other aspects. This structure is crucial for a shape's appearance. If we want to synthesize shapes automatically, we have to take structural properties into account in order to create high-quality results. Procedural modeling techniques explicitly define structure with a set of rules. While the results can be remarkably good, it requires deep knowledge about the desired design principles and sophisticated skills in programming these rules. We seek to provide a more natural way to construct new shapes: Given an example shape, our goal is to derive a shape grammar that produces shapes similar to the exemplar. The basic principle of example based modeling presents an intuitive interface for artists since they only need to create shapes.

Existing example-based methods have strong limitations that restrict their use to specific scenarios only. Methods that are based on non-parametric texture synthesis consider geometry synthesis in the sense of geometric texturing: given a smooth surface and some example geometry, the method tries to add more detail to the smooth shape where the geometric details appears similar to those of the exemplar [Bhat et al., 2004; Sharf et al., 2004; Lai et al., 2005; Nguyen et al., 2005; Zhou et al., 2006; Zelinka and Garland, 2006; Chen and Meng, 2009]. We, however, would like to generate shapes without the need of specifying a coarse target space. A fundamental problem of synthesizing 3D surfaces is rooted in the nature of surfaces: they are 2d elements embedded in 3D space. This means that it is not sufficient to transfer the idea of texture synthesis to geometry because the domain is not known in that the MRF optimization should happen. In 2d texture synthesis the goal is to find a color value for each pixel with the objective of consistent neighborhoods. Correspondingly, in geometric texture synthesis most methods define a



(a) input model *(b) symmetric area* *(c) docking sites* *(d) result*

(e) input model *(f) symmetric area* *(g) delete* *(h) insert*

Figure 5.1: *(a) We compute shape modification operations by examining the partial symmetry structure of a 3D model. (b) Symmetric regions are marked in red. (c) A set of symmetric curves that cuts the model into two pieces yields a docking site that corresponds to (d) a replacement operation. (f) Symmetric regions for a different transformation T (red region maps to blue region under T) yields deletion(g) and insertion (h).*

smooth 2d surface embedded in 3D space and synthesize geometric details for each surface element with geometric consistent neighborhoods. If we want to create new shapes from scratch, we need to optimize both: geometric consistency of neighboring elements *and* the domain in which the shape lives. Some methods bypass this problem by considering 3D texture synthesis on a regular grid [Bhat et al., 2004; Merrell, 2007; Merrell and Manocha, 2008] with the drawback of being limited to a specific grid structure. So far, no method based on texture synthesis has been proposed that is able to create 3D shapes from scratch with results comparable to those presented in this chapter.

In our approach, we use the idea of consistent neighborhoods in non-parametric texture synthesis [Efros and Leung, 1999]. However, we formulate a more restrictive criterion for our output models: we demand that any point on the resulting shape matches some point on the exemplar within a local neighborhood of radius r (we call this criterion r-*similarity*). As we will see in the results, the strict enforcement of r-similarity yields plausible results even for highly structured shapes. The reason for this can be explained by the interplay of r-similarity and the exemplars structure: r-similarity restricts the potential number of neighboring elements to a subset that preserves geometric structures locally. Further, r-similarity induces that every part of the new shape has been observed in the exemplar, also including shape boundaries. Consequently, we cannot create "open" surfaces without these observed boundaries and forces us to assemble globally consistent shapes. This combination of local and global consistency leads to a restricted set of possible

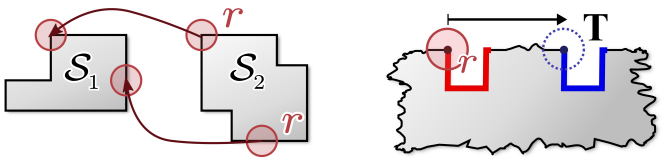
shapes that preserve many structural properties of the exemplar.

In contrast to texture synthesis, our approach constructs a set of explicit, procedural rules that encode how to build such objects efficiently, leading to an *inverse procedural modeling* system [Aliaga et al., 2007; Šťava et al., 2010]. The key observation of this work is that the partial symmetries of an object reveal a set of *shape operations* that alter the object while guaranteeing strict r-similarity: Assume we fix a transformation $\mathbf{T}$ and look at the regions of an object (Figure 5.1a) that are symmetric under this transformation (Figure 5.1b). We might find non-symmetric regions that are separated from the rest of the model by some symmetric area. We call these regions *dockers* and a curve through a symmetric region that cuts out a docker a *docking site* (Figure 5.1c). As the docking site geometry matches, we can exchange the corresponding dockers while maintaining similarity to the input exemplar. In general, we obtain operations that insert, delete, or replace pieces of $\mathcal{S}$ (Figure 5.1d,g,h). We examine the dependency of such operations and encode the result in a *shape grammar* that encodes a set of r-similar objects. We consider three variants: general rewriting systems, context-free grammars, and supplemental grid structured production rules. We implement the described analysis framework in a numerically robust way, handling general triangle meshes as well as point cloud data from 3D scanners, and demonstrate prototypical tools for shape modeling by example.

Our technique is the first that is able to compute shape grammars for general 3D surfaces from example geometry without any user interaction (the similarity radius r is the only parameter). We provide a theoretical framework that establishes an interesting link between partial symmetry of an object $\mathcal{S}$ and a space of objects similar to $\mathcal{S}$. It provides strict formal guarantees: All computed models are strictly r-similar to the exemplar. In particular, any topologically consistent, closed input manifold will yield output models with these properties. However, in order to provide such strict guarantees, our formal framework requires models that have perfect partial symmetries, which is a main limitation of the presented approach. This could easily be generalized to different notions of symmetry and similarity.

5.1 Formal Model

In this section, we present our theoretical framework, proceeding in the following steps: First, we define the basic notions of *symmetry* and *similarity* (Subsection 5.1.1). Then, we describe how to compute *docking sites*, *dockers*, and how to construct the associated *shape operations* (Subsection 5.1.2). Afterwards, we bring these elements into a canonical form in Subsection 5.1.3. Finally, we discuss how to combine the obtained rules into a *shape grammar* (Subsection 5.1.4).



(a) r-similarity *(b) r-symmetry*

Figure 5.2: *(a) r-similarity: every point of $\mathcal{S}_2$ is locally similar to a point of $\mathcal{S}_1$, within a radius of r. (b) r-symmetry is defined analogously: under a fixed transformation $\mathbf{T}$, symmetric points must be similar within a radius of r.*

5.1.1 Similarity and Symmetry

Input and neighborhoods

We assume that we are given an *input exemplar* $\mathcal{S} \subset \mathbb{R}^3$. For simplicity, we assume a finite, piecewise smooth surface. For every $\mathbf{x} \in \mathcal{S}$, we define the *$r$-neighborhood* $N_r^{\mathcal{S}}(\mathbf{x}) = N_r(\mathbf{x}) := \{\mathbf{y} \in \mathcal{S} \mid dist(\mathbf{x}, \mathbf{y}) \leq r\}$, where $dist$ is a metric on $\mathcal{S}$. In the following, we use the intrinsic distance within $\mathcal{S}$ to correctly handle close pieces that are topologically disconnected.

Transformations

In the following, symmetry and similarity are defined with respect to a *group of admissible transformations* $\mathcal{T}^+$ consisting of continuous, injective functions $\mathbf{T} : \mathcal{S} \to \mathbb{R}^3$. We denote the *transformation of a set* $\mathcal{A}$ as $\mathbf{T}(\mathcal{A}) := \{\mathbf{T}(\mathbf{x}) \mid \mathbf{x} \in \mathcal{A}\}$. Here, we restrict ourselves to the group of rigid motions. However, most of our framework could be easily generalized to different notions of symmetry and similarity, induced for example by invertible affine mappings or by intrinsic isometries.

We can now define our notion of local similarity of objects: An object is similar to an exemplar if each local neighborhood of the object can be found somewhere in the exemplar (Figure 5.2a). We formalize this concept in the notion of r-similarity.

r-similarity

Given two surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$, $\mathcal{S}_2$ is *r-similar* to $\mathcal{S}_1$ if and only if for all $\mathbf{y} \in \mathcal{S}_2$ there exist a point $\mathbf{x} \in \mathcal{S}_1$ and a mapping $\mathbf{T} \in \mathcal{T}^+$ such that:

(1) $\mathbf{T}(\mathbf{x}) = \mathbf{y}$ and $\mathbf{T}(N_r^{\mathcal{S}_1}(\mathbf{x})) \subseteq \mathcal{S}_2$.

(2) For all $\tilde{\mathbf{x}} \in N_r^{\mathcal{S}_1}(\mathbf{x})$: $\mathbf{x} \cong \mathbf{T}(\tilde{\mathbf{x}})$

where "$\cong$" means that the local topology is preserved with respect to the surfaces $\mathcal{S}_1$ and $\mathcal{S}_2$ where the points are contained in (formally: there exist infinitesimal neighborhoods that are homeomorphic). Condition (1) ensures that we find a geometrically matching neighborhood in $\mathcal{S}_1$ for every point of $\mathcal{S}_2$. Condition (2) makes

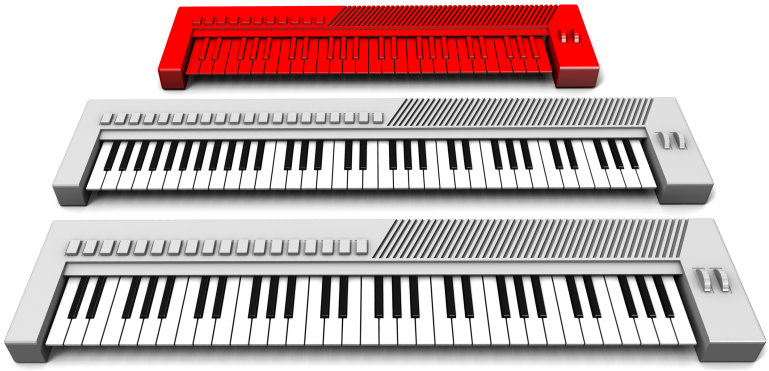


Figure 5.3: *Effect of parameter r: Smaller values of r lead to a larger number of possible shape operations that allows us to insert single piano keys (middle row). A larger value of r enforces a correct piano key pattern, where we can only add or delete an entire octave (bottom row). The original model is shown in red (top row).*

sure that the topology matches as well. In particular, we cannot map a manifold interior point to a boundary point, or an interior point to a T-junction, and vice versa.

Please note that the definition of r-similarity is not symmetric. We always place the r-neighborhood on the exemplar surface $\mathcal{S}_1$, never on the surface that $\mathbf{T}$ maps to. This is important for generalizations to non-rigid mappings where $\mathbf{T}$ might contains scaling that alters the size of the neighborhood. Analogous to r-similarity, we define the notion of r-symmetry, where we map neighborhoods within a single surface. We start by defining infinitesimal symmetry:

Symmetry under a transformation $\mathbf{T}$

We denote the set of all points of $\mathcal{S}$ that are symmetric with respect to a transformation $\mathbf{T}$ as:

$$\xi(\mathbf{T}) := \{\mathbf{x} \in \mathcal{S} \mid \mathbf{T}(\mathbf{x}) \in \mathcal{S} \text{ and } \mathbf{x} \cong \mathbf{T}(\mathbf{x})\}\,,$$

Two points of $\mathcal{S}$ are symmetric under a transformation $\mathbf{T}$ if $\mathbf{T}$ maps between them and their infinitesimal local neighborhood is topologically equivalent. Accordingly, the non-symmetric points with respect to $\mathbf{T}$ are denoted by $\overline{\xi}(\mathbf{T}) := \mathcal{S} \backslash \xi(\mathbf{T})$.

r-symmetry under a transformation $\mathbf{T}$

The set of all points that are *r-symmetric* under a transformation $\mathbf{T}$ is given by:

$$\xi_r(\mathbf{T}) := \{\mathbf{x} \in \mathcal{S} \mid \text{all } \tilde{\mathbf{x}} \in N_r(\mathbf{x}) \text{ are symmetric under } \mathbf{T}\}$$

This means, a point is r-symmetric iff its whole r-neighborhood is symmetric under the same transformation (Figure 5.2b). In other words, we obtain the r-symmetric points by an erosion operation of radius r on the set of points that map

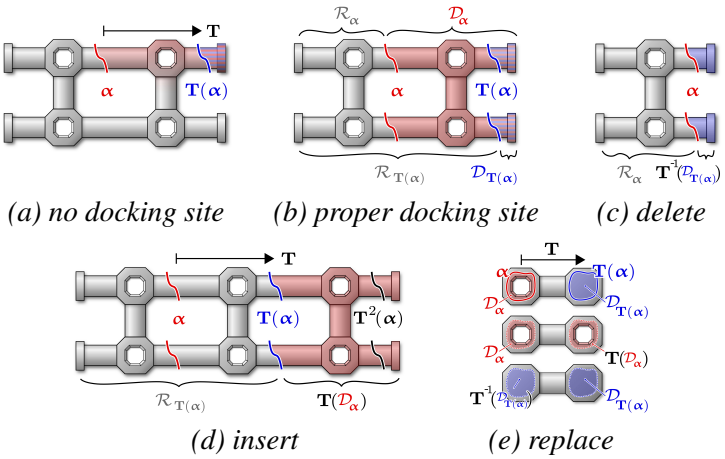


Figure 5.4: *Definition of a docking site. (a) The first part of the definition is met, but the docking site does not divide the model into two disconnected pieces. Therefore, no valid operation results. (b) A valid docking site, meeting all three criteria. (c),(d) Dockers can be exchanged while keeping the model r-similar to $\mathcal{S}$. If primary and secondary dockers are contained in each other, this results in inserting or deleting symmetric pieces. (e) Otherwise, geometry is just replaced; please note that this example shows a different geometry.*

from $\mathcal{S}$ back to $\mathcal{S}$ under $\mathbf{T}$. We will use this observation in Section 5.2 to devise a simple and efficient algorithm for computing the r-similarity structure of an object. The algorithm will actually work on a discrete set of candidate transformations $\mathbf{T}$, constructed by matching surface features, for which it computes the symmetric area. For now, we will just assume that we know $\xi_r(\mathbf{T})$ for all transformations $\mathbf{T}$ that might be of interest and discuss the details of symmetry detection later.

5.1.2 Dockers, Docking Sites and Shape Operations

Our goal is now to determine shape operations that modify an exemplar shape while maintaining r-similarity to the original. The key insight of this chapter is that the symmetry structure of an object reveals a rich set of such operations as well as their interdependence. For now, we will consider one fixed transformation $\mathbf{T}$ and examine the r-symmetry structure of $\mathcal{S}$ with respect to $\mathbf{T}$. How to combine the result from different transformations within a shape grammar will be discussed later in Subsection 5.1.4.

$\xi_r(\mathbf{T})$ can be regarded as a binary function on the exemplar $\mathcal{S}$ that marks symmetric regions on the surface (Figure 5.1b shows an example visualizing $\xi_r(\mathbf{T})$ as red area). Obviously, any subset $\mathcal{X} \subseteq \xi_r(\mathbf{T})$ of a symmetric region can be exchanged with its corresponding geometry $\mathbf{T}(\mathcal{X})$ without changing the geometry at all. However, if we find a piece of non-symmetric geometry that is divided from the rest of the model by symmetric area, we can use the symmetric region as

a *docking site* to replace some geometry with different geometry while maintaining r-similarity to the exemplar. We formalize this observation in the following definition:

Docking sites and dockers

A set of surface curves $\boldsymbol{\alpha} \subseteq \mathcal{S}$ is called a *docking site* with respect to a transformation $\mathbf{T}$, if the following three properties hold (see Figure 5.4a,b):

- $\boldsymbol{\alpha}$ is r-symmetric under $\mathbf{T}$: $\boldsymbol{\alpha} \subseteq \xi_r(\mathbf{T})$.
- $\boldsymbol{\alpha}$ partitions the model into two topologically disconnected pieces $\mathcal{D}_{\boldsymbol{\alpha}}$ and $\mathcal{R}_{\boldsymbol{\alpha}}$. This means, any continuous path in $\mathcal{S}$ between the two pieces must intersect $\boldsymbol{\alpha}$. The piece $\mathcal{D}_{\boldsymbol{\alpha}}$ contains geometry that is not symmetric under $\mathbf{T}$: $\mathcal{D}_{\boldsymbol{\alpha}} \not\subseteq \xi_r(\mathbf{T})$. We call $\mathcal{D}_{\boldsymbol{\alpha}}$ a *docker* for *docking site* $\boldsymbol{\alpha}$
- $\mathbf{T}(\boldsymbol{\alpha})$ also partitions the model into two topologically disconnected pieces $\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$ and $\mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})}$. We call $\mathbf{T}(\boldsymbol{\alpha})$ the *secondary docking site* and $\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$ the *secondary docker* of $\boldsymbol{\alpha}$.

For clarity we will sometimes refer to the original, non-transformed docking sites (and dockers) as *primary* docking sites (and *primary* dockers), as opposed to secondary docking sites. By convention, the docking site $\boldsymbol{\alpha}$, the docker $\mathcal{D}_{\boldsymbol{\alpha}}$ and the remaining geometry $\mathcal{R}_{\boldsymbol{\alpha}}$ are defined to be disjoint. The same applies to the secondaries. It is important to note that the curves that form the docking sites are in general not restricted to a single connected component. Figure 5.4b shows such an example. In addition, for input manifolds without boundary, the docking sites always have to be a collection of closed curves (if this was not the case, the docking sites could not partition the model into disconnected pieces). If the input exemplar $\mathcal{S}$ contains boundaries, the docking sites can potentially contain open curves.

Figure 5.4b shows an example of a partitioning of the model into the pieces $\mathcal{D}_{\boldsymbol{\alpha}}$, $\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$, $\mathcal{R}_{\boldsymbol{\alpha}}$, and $\mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})}$. Obviously, this is only possible if $\boldsymbol{\alpha}$ and $\mathbf{T}(\boldsymbol{\alpha})$ cut the model into disconnected pieces (Figure 5.4a). Figure 5.4c,d show how we can use docking sites and dockers to modify geometry: The docking site itself is situated within symmetric geometry. Therefore, we can cut the model through the docking sites and replace the docker with the secondary docker, or vice versa. By construction, the boundary is r-similar so that the pieces fit together. Formally, every shape operation is performed by the same rule, a *replacement* of dockers.

Replacement shape operations

Let $\boldsymbol{\alpha}$ be a docking site with respect to transformation $\mathbf{T}$. A *shape operation*, $\mathtt{op}_{\boldsymbol{\alpha},\mathbf{T}}$ is given by

$$\mathtt{op}_{\boldsymbol{\alpha},\mathbf{T}} : \mathcal{S} \to \mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})} \cup \mathbf{T}(\mathcal{D}_{\boldsymbol{\alpha}}) \cup \mathbf{T}(\boldsymbol{\alpha}).$$

It replaces the secondary docker with the transformed primary docker (Figure 5.4c and Figure 5.4e, middle row). Correspondingly, there is also a *secondary shape operation* $\mathtt{op}_{\mathbf{T}(\boldsymbol{\alpha}),\mathbf{T}^{-1}}$

$$\mathtt{op}_{\mathbf{T}(\boldsymbol{\alpha}),\mathbf{T}^{-1}} : \mathcal{S} \rightarrow \mathcal{R}_{\boldsymbol{\alpha}} \cup \mathbf{T}^{-1}(\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}) \cup \boldsymbol{\alpha},$$

which replaces the secondary docker with the primary docker, transformed accordingly (see Figure 5.4d and Figure 5.4e, lower row).

Obviously, a secondary operation for a docking site $\boldsymbol{\alpha}$ and transformation $\mathbf{T}$ is identical to the primary replacement operation for the docking site $\mathbf{T}(\boldsymbol{\alpha})$ and transformation $\mathbf{T}^{-1}$. Please note that our notation for the shape operations already makes use of this observation. The consequence in practice is that we can restrict ourselves to collecting only primary operations; we will obtain the secondary operations automatically by considering the corresponding symmetries under the inverse transformation.

Classification

We can distinguish further between different effects of the primary operations by looking at the topological relation of primary and secondary dockers. We obtain three cases that lead to different types of operations, as illustrated in Figure 5.4c-e:

- **Insert:** $\mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})} \subset \mathcal{D}_{\boldsymbol{\alpha}}$. The secondary docker is a subset of the primary docker (Figure 5.4c).
- **Delete:** $\mathcal{D}_{\boldsymbol{\alpha}} \subset \mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})}$. The primary docker is a subset of the secondary docker (Figure 5.4d).
- **Replace:** All other cases (Figure 5.4e).

Replace operations just substitute parts of the geometry with an alternative piece of geometry. Insert and delete operations are special cases of replacements that replace a piece of geometry with two or zero copies of itself so that they effectively grow or shrink the object. These operations (and combinations of them) are useful for "smart resizing" of 3D objects; we will make use of this later in the applications section. Please note that the replace shape operation is still possible if the dockers overlap partially, i.e. $\mathcal{D}_{\boldsymbol{\alpha}} \cap \mathcal{D}_{\mathbf{T}(\boldsymbol{\alpha})} \neq \emptyset$. However, the operations become dependent (which also frequently occurs for operations derived from two different transformations). Resolving dependencies is discussed in Subsection 5.1.4.

Collision avoidance

An important aspect we neglect so far is that of *collisions*. We have to make sure that $\mathbf{T}(\mathcal{D}_{\boldsymbol{\alpha}})$ and $\mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})}$ do not *collide*, i.e., have a non-zero distance everywhere, except from the docking site where they meet. If this is not the case but the a newly inserted piece collides with already existing geometry, the operation might not be

valid and we do not perform the shape operation. We can now formulate the main result of this subsection:

Maintaining r-similarity

Given an input surface $\mathcal{S} \subset \mathbb{R}^3$ and an r-docking site $\boldsymbol{\alpha}$ under transformation $\mathbf{T}$. If the shape operation $\mathtt{op}_{\boldsymbol{\alpha},\mathbf{T}}$ is not colliding, it will create a result that is r-similar to $\mathcal{S}$.

This is easy to see if we take a closer look at the individual components: The remaining geometry $\mathcal{A} := \mathcal{R}_{\mathbf{T}(\boldsymbol{\alpha})} \cup \mathbf{T}(\boldsymbol{\alpha})$ and the inserted docker $\mathcal{B} := \mathbf{T}(\mathcal{D}_{\boldsymbol{\alpha}})$ that are assembled by the shape operations are parts of $\mathcal{S}$. Therefore they are trivially r-similar to $\mathcal{S}$ everywhere, except from the region that has a distance of less than r to the cutting curve $\mathbf{T}(\boldsymbol{\alpha})$. For the points in a band of distance r to both sides of the cutting line $\mathbf{T}(\boldsymbol{\alpha})$, we know that the geometry is r-symmetric. This means that the geometry in this area is identical for both pieces $\mathcal{A}$ and $\mathcal{B}$. Therefore, any point in this band is r-similar as well: for points from $\mathcal{A}$, we can find an r-neighborhood in piece $\mathcal{A}$ under an identity transformation; for pieces from $\mathcal{B}$, we find an r-neighborhood in $\mathcal{D}_{\boldsymbol{\alpha}} \subseteq \mathcal{S}$ under transformation $\mathbf{T}^{-1}$. Because we avoid collisions, we also cannot change the local geometry indirectly through intersections of distant pieces.

5.1.3 Elementary Docking Sites

So far, we have shown how to extract shape operations from the symmetry structure of an exemplar surface. However, there are a few technical problems left: First, we can get an infinite number of equivalent operations by moving the docking site within the symmetric region (Figure 5.5a). We thus need to normalize the construction so that we get a canonical representation for a shape operation. Even after doing this, the number of shape operations for a single transformation $\mathbf{T}$ might still be exponential in the number of non-symmetric "docker" regions, because the docking site can select an arbitrary subset of such regions that are located within a common symmetric region (Figure 5.5b). This obviously prevents us from compactly encoding the set of all discovered shape operations. *Elementary docking sites* (Figure 5.5c) address both problems:

Elementary docking sites

An *elementary docking site* is a docking site for which the closure of its docker is minimal with respect to set inclusion. Shape operations derived from elementary docking sites are called *elementary shape operations*.

Intuitively, we obtain an elementary docking site by shrinking the docker enclosed by the docking site as much as possible, enclosing only the non-symmetric region, at the boundary between r-symmetric and non-r-symmetric area (with respect to a certain transformation $\mathbf{T}$). In addition, we are not allowed to enclose

more non-r-symmetric regions than necessary to divide the model into two pieces, because otherwise, the docker would not be minimal. We now discuss how to compute elementary docking sites. For this, and all of the following, we will always assume that we have a finite number of non-symmetric regions, i.e., $\overline{\xi_r(\mathbf{T})}$ consists of a finite number of connected sets, each with continuous boundary curves. For practical models such as triangle meshes, this assumption is always met. The idea for the algorithm is rather simple: For any valid docker, we need to make sure that a docker is separated completely from the remaining geometry by the docking sites, which can be solved by simple region growing. However, we need to simultaneously assure that the same is true for the corresponding secondary dockers, which therefore requires alternated region growing in the primary and secondary domain.

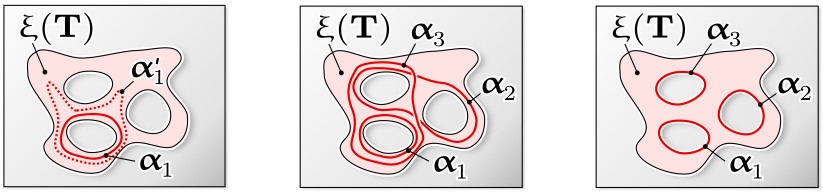
Algorithm: Computing elementary docking sites

Input: We are given an input model $\mathcal{S}$, a transformation $\mathbf{T}$, and the symmetry structure $\xi_r(\mathbf{T})$.
Algorithm: We start by computing the boundary curves $\partial\overline{\xi_r(\mathbf{T})}$ of the non-symmetric regions using region growing, as well as the secondary boundaries $\mathbf{T}(\partial\overline{\xi_r(\mathbf{T})})$, which must exist due to symmetry. This partitions the model into a set of primary and secondary non-symmetric regions, which will be combined to valid and elementary primary and secondary dockers next. We build a graph that encodes the dependencies of these regions: We connect primary and secondary non-symmetric regions if they share a boundary curve where the primary boundary maps to the secondary under $\mathbf{T}$. The important observation is that this does not need to be symmetric. For example, the secondary can map to two boundaries while the primaries only map to one each, as shown in Figure 5.6. In order to obtain a valid combination we compute the connected components in this dependency graph. All primary regions within a connected component form one primary docker, and all secondary regions the secondary docker. Their boundaries to the symmetric area are the docking sites. These pieces are elementary because no smaller set can cut the model into two pieces on both primary and secondary side. Figure 5.6 shows how we subsequently add dependent pieces to extract a valid pair of primary/secondary dockers.

One can easily show that elementary dockers for the same transformation $\mathbf{T}$ are always disjoint. Because of this, it is easy to see that any general shape operation with respect to a fixed transformation $\mathbf{T}$ can always be achieved by a combination of elementary shape operations. This means, for n different non-symmetric regions, we only need to store $\mathcal{O}(n)$ elementary shape operations, rather than in the worst case $2^n - 1$ non-elementary ones.

5.1.4 Extracting Shape Grammars

So far, we have found a number of operations that allow us to change the input shape $\mathcal{S}$ while keeping it r-similar to the original. However, we can only change



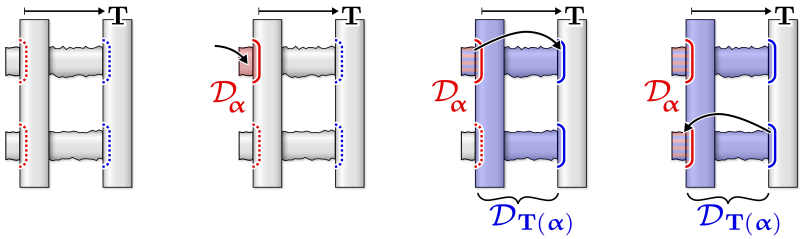
(a) equivalent sites *(b) combinatorial explosion* *(c) elementary sites*

Figure 5.5: *General vs. elementary docking sites: (a) The docking sites α_1 and α_1' are equivalent. (b) By combining multiple non-symmetric regions, we can obtain an exponential number of non-equivalent docking sites. (c) Elementary docking sites form a canonical set for composing more complex operations.*

the original input once: the changes might alter the situation such that the original operations are not applicable anymore. In order to enable the execution of multiple, consecutive shape operations, we therefore need to to determine their interdependency, and if necessary adapt the executed operation accordingly. In the following, we will describe three different models that build sets of rules that combine multiple shape operations. We start with a straightforward construction that yields a general Chomsky type-0 grammar (general rewriting system) based on shape matching. This model is the most expressive, but its structure is the least computationally accessible. Therefore, in the next step, we compute a context-free subset of this grammar, which is easier to handle in applications. Lastly, we add non-context free grid-based replication rules with multiple degrees of freedom, which are very useful for analyzing real-world objects [Müller et al., 2007].

A Basic Shape Matching Grammar

Assume that we are given a set of shape operations for an exemplar $\mathcal{S}$ and we want to execute multiple of these operations subsequently while maintaining r-similarity to $\mathcal{S}$. The important observation at this point is that we can apply a shape operation in any place where we find a suitable docking site. "Suitable" means that the new docking site is related to the original one by an admissible transformation $\mathbf{T} \in \mathcal{T}^+$. In that case, we can just transform the docker of the shape operation by $\mathbf{T}$ to match the new docking site. As the admissible transformations form a group by definition, the compound transformation will automatically be admissible as well and the transformed operation still guarantees r-similarity. This directly yields a shape matching grammar: Before each operation, we have to search the model for matching docking sites (using rigid shape matching, in our case) and then obtain a selection of applicable operations. Obviously, this approach is costly to compute and incompatible with standard procedural modeling tools. In addition, the structure of the language might be very complex. As a general rewriting system (Chomsky-0 grammar), most properties of the language are uncomputable and thus inaccessible for controlling the modeling process. Therefore, we have to extract a



(a) symm. borders *(b) no docker yet* *(c) dep. secondary* *(d) final output*

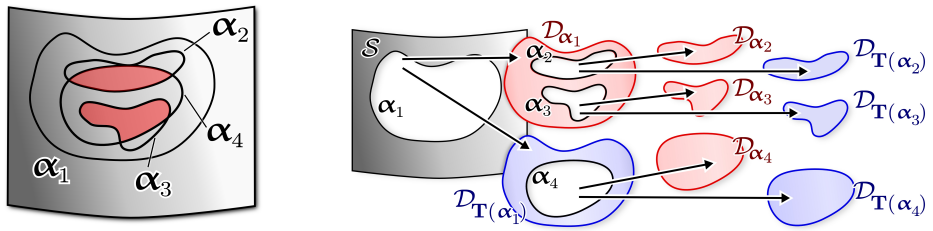
Figure 5.6: *Computing elementary docking sites: (a) Transformation* $\mathbf{T}$ *and corresponding boundaries of* r*-symmetry. (b) The red piece is non-symmetric. (c) It is not a docker (yet) because its secondary boundary does not split the model. (d) Adding the missing dependent piece yields valid primary/secondary dockers.*

more manageable subset. This is the subject of the next subsection:

Context Free Grammars

In a context free grammar, we do not perform shape matching but use *non-terminal symbols* to identify a space where further geometry can be plugged in. The geometry itself is encoded as *terminal symbols*. A set of *production rules* describes which terminal and non-terminal pieces can be plugged into the space designated by each non-terminal piece, leading to a hierarchical structure of insertions of pieces. In our framework, *docking sites* represent non-terminal symbols because alternative geometry can be plugged into the regions they enclose (Figure 5.7). *Dockers* are the pieces being plugged in, which in turn might contain further sub-docking-sites. Therefore, the geometry of a docker, minus that enclosed by sub-docking-sites, corresponds to terminal symbols, and the contained docking sites are hierarchically dependent non-terminals. We will reflect this fact in our notation and use non-terminals and docking sites interchangeably, as well as dockers and terminal symbols.

Using context-free production rules means that we need to be able to tell a priori whether a certain piece of geometry that is tagged with a non-terminal node will allow for the insertion of certain alternative pieces or not. We cannot retract from this decision later on because we do not perform any online shape matching. Any conflicts between rules that try to alter the same piece of geometry must be already resolved during the construction of the formal language. Obviously, potential conflicts are only created by designating docking sites. Identifying a docker and storing its terminal geometry cannot create conflicts, no matter how these dockers overlap on the exemplar. However, if we put a docking site on a piece of geometry, we must make sure that the docker area it encloses does not intersect with further docking sites, because the geometry within these bounds is subject to change. More precisely, if we create the production rule for a single docker, the docking sites (i.e., the non-terminals) within this rule must not overlap. We use the following formal



a) b)

Figure 5.7: *Overlapping docking sites cannot be used at the same time. Resolving these conflicts and extracting a hierarchy of docking sites and dockers yields a context free grammar.*

classification: Two docking sites $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ are *conflicting* iff $\boldsymbol{\alpha}_1 \cap \boldsymbol{\alpha}_2 \neq \emptyset$, i.e., if the *docking sites* intersect. $\boldsymbol{\alpha}_2$ is *hierarchically dependent* on $\boldsymbol{\alpha}_1$ iff $\mathcal{D}_{\boldsymbol{\alpha}_2} \subset \mathcal{D}_{\boldsymbol{\alpha}_1}$, i.e., the *dockers* are contained in each other. Obviously, hierarchically dependent docking sites are not conflicting in the sense of this definition, but we still cannot combine the rules arbitrarily but need to make sure that the docker they are based on does already exist. This idea leads to a hierarchical construction algorithm. The main idea is to first identify hierarchical dependence of docking sites. Then we iteratively consider each docker and handle all docking sites it directly contains. If these docking sites are conflicting, we remove some of them until all conflicts are resolved. As there are multiple alternative ways to achieve this, we in general obtain several production rules for this docker. We now look at this more in detail:

Constructing a context-free shape grammar: We first build a tree that encodes the hierarchical dependence of the docking sites. A node $\boldsymbol{\alpha}$ is contained in the subtree of a node $\boldsymbol{\beta}$ if and only if $\boldsymbol{\alpha}$ is completely contained in $\mathcal{D}_{\boldsymbol{\beta}}$. This always yields a unique tree because the inclusion relation is tree structured by definition. After that, we look at each node in the tree from bottom up, starting at the leaf nodes. The docker of each leaf node yields a terminal node in our grammar that contains only fixed geometry and no further non-terminals. The docking site of each leave nodes is represented as a non-terminal node and we create the corresponding production rules that encode two alternatives each: inserting either the primary or the secondary docker. Now we go up one level in the hierarchy and build more complex rules: Let $\boldsymbol{\alpha}$ be an inner node, and $\mathcal{C}(\boldsymbol{\alpha})$ be the set of all docking sites that are direct children of this node. These children form areas where we can insert hierarchically dependent pieces (which in turn might contain further docking sites, i.e., further non-terminals). However, the docking sites in $\mathcal{C}(\boldsymbol{\alpha})$ will in general overlap, creating conflicts. Therefore, we create multiple alternative rules that each resolve the conflicts in a different way: We consider the graph of overlapping docking sites, where the docking sites in $\mathcal{C}(\boldsymbol{\alpha})$ are the vertices and an edge is inserted if the two sites are conflicting. In this graph, we compute the set of all maximal independent sets. This yields the set of all sets of docking sites that do not overlap each other. In order not to miss options, we also need to unfold the hierarchy of overlapping

nodes and include these in the independent set computation. Each of the obtained independent sets $\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_k\}$ is free of collisions and is therefore converted into one production rule for the non-terminal $\boldsymbol{\alpha}_0$:

$$\boldsymbol{\alpha} \rightarrow \underbrace{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_k}_{\text{non-terminals}}, \quad \underbrace{\left(\mathcal{D}_{\boldsymbol{\alpha}} \setminus (\mathcal{D}_{\boldsymbol{\beta}_1} \ldots \mathcal{D}_{\boldsymbol{\beta}_k})\right)}_{\text{terminal remainder geometry}}$$

The production rule encodes that the non-terminal $\boldsymbol{\alpha}$ can be replaced by a set of non-conflicting docking sites and the remaining geometry not covered by these sites. For each independent set, one such separate rule is created. Once we have performed the operation for the primary docker of $\boldsymbol{\alpha}$, we repeat the same procedure for the secondary docker, which also fits into $\boldsymbol{\alpha}$.

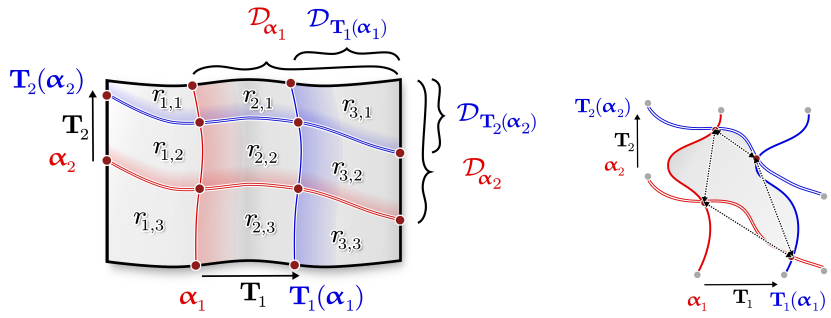
Complexity problem: The strategy has a problem in practice: the number of maximal independent sets in a graph has an exponential worst-case lower bound of $\Omega(3^{n/3})$ sets for n nodes. Correspondingly, the algorithm might become impractical for complex inputs. Therefore, we use a bounded complexity approximation in practice: Instead of enumerating all independent sets, we sample the solution space randomly. We iteratively choose a random node and remove all colliding nodes until no more nodes are left. We make sure to start at a different node each time and limit the number of trials to a fixed constant (in our examples: 10). Thus the maximum number of computed production rules is fixed as well. In addition, we only unfold at most one hierarchy level; if such a node still collides, we dismiss all hierarchically contained docking sites as well. This approximation does not yield the largest possible context free sub-language but it is very fast and produces good results in practice.

Improvements: In order to make the grammar more expressive, we perform shape matching between docking sites before constructing the grammar and identify all docking sites of the same shape. This means that not just the primary and secondary docker can be inserted into the corresponding non-terminal symbol but all dockers with similar docking site. In addition, we also try to avoid docking sites that are created by continuous symmetries, such as a window that can slide across a flat wall. We remove such rules by detecting slippable docking sites using the technique of [Gelfand and Guibas, 2004]. We use this optional filter in all of our examples.

Regular Grids

Many real-world objects contain grid structures, such as a grid of $n_1 \times n_2$ windows in the facade of a building. In the following, we will call such structures k-grids, were k is the number of discrete degrees of freedom. A context free grammar cannot represent k-grids for $k \geq 2$ (unless we fix the repetition counts a priori, which is not useful in our application). Thus, we add a separate grid replication rule to our shape grammar that models this case.

1-grids: In our framework, 1-grids are identified by "insert" and "delete" shape operations: By definition, these operations are always dual to each other, and they



(a) configuration for a grid patterns *(b) no valid grid*

Figure 5.8: *(a) Two colliding insert/delete operations divide the model into regions of nine different types. The corners are used once, the central piece is repeated on a 2-grid, and the rest is linearly repeated. (b) Counterexample: This arrangement of docking sites does not form a tileable grid cell.*

mutually undo the effect of each other. In addition, the insert operation can be repeated an arbitrary number of times (up to collisions) because the inserted part by construction contains a docking site for another insertion.

2-grids: Grids with more than one degree of freedom show up as collisions of docking sites of multiple 1-grids. We first consider the case $k = 2$ for two insert type shape operations $\mathtt{op}_{\boldsymbol{\alpha}_1,\mathbf{T}_1}, \mathtt{op}_{\boldsymbol{\alpha}_2,\mathbf{T}_2}$. If the docking sites $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ intersect, the corresponding primary and secondary dockers intersect as well, due to symmetry. We classify the pieces by up to nine possible cases (Figure 5.8a). We choose an index of 1 for the remainder geometry, 2 for within the docking site but outside the secondary docker and 3 for inside the primary and the secondary docker. Accordingly, we label the pieces by $r_{1,1}$ up to $r_{3,3}$, taking both operations into account. Please note that several disconnected pieces of each type might exist because the exemplar $\mathcal{S}$ can be of arbitrary topology and some types of pieces might be missing altogether. Pieces of different type have a different purpose: types $r_{1,1}, r_{1,3}, r_{3,1}$, and $r_{3,3}$ form the "corner stones" that are instantiated exactly once. The pieces $r_{1,2}, r_{2,1}, r_{2,3}, r_{3,2}$ are replicated in one direction each, forming the boundary of the grid. Finally, the $r_{2,2}$ pieces are replicated in two directions.

Identifying tileable grids: Not all pairs of conflicting 1-grids create feasible 2-grids because the inner regions $r_{2,2}$ can have non-tileable boundaries. This can happen because a pair of symmetric primary/secondary docking sites is cut into pieces by a different pair of docking sites. Although the pairs of curves are symmetric, the intersections do not need to be symmetric (Figure 5.8b). In order to obtain a valid tileable grid, we must demand two additional properties: First, the boundary curves of all regions $r_{i,j}$ must be symmetric under the transformations $\mathbf{T}_1, \mathbf{T}_2$. Second, the transformations $\mathbf{T}_1, \mathbf{T}_2$ must commute to obtain a well defined grid [Pauly et al., 2008]. If these two conditions are met, and we additionally assume that the inserted pieces do not collide with each other, it is easy to see

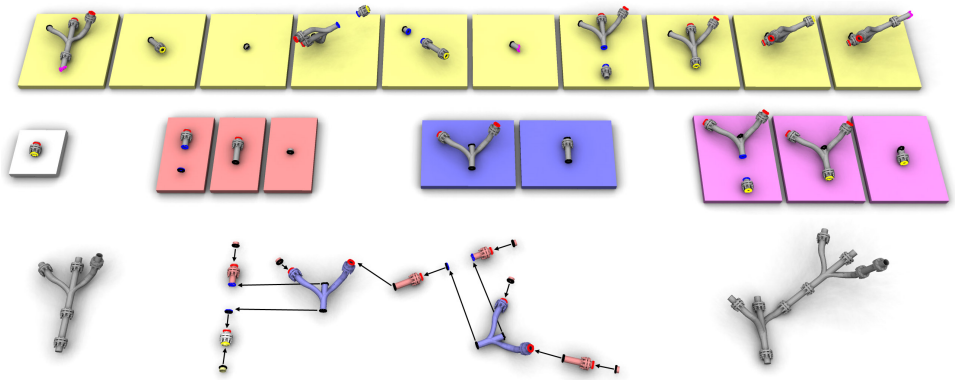


Figure 5.9: *Visualization of the grammar computed for the "pipe tree" example (Figure 5.10). Each pad carries a docker, with the color indicating matching docking sites, shown as curves in the same color. Black curves indicate the sites at which the dockers are inserted into the parent docking site. The white docker is the root of the grammar. Below, an example assembly is given for illustration.*

that any instance of the grid is r-similar to the input surface. Our implementation currently checks all these conditions explicitly.

General case: The grid construction generalizes to the case of grids with k degrees of freedom: All simplices consisting of 2^k points need to match each other under the corresponding transformations, and all transformations need to commute. We can incorporate grids into the shape grammar by applying the grid detection algorithm to conflicting docking sites of the hierarchically extracted dockers.

5.2 Implementation

It is possible to perform all the computations described in the previous section directly on triangle meshes. However, this easily leads to robustness problems in practice due to sliver triangles. We avoid these issues by using an approximate representation: We store the symmetry information $\xi_r(\mathbf{T})$ only up to a fixed sampling resolution ϵ_s, in a voxel grid with uniform spacing ϵ_s. Each voxel that intersects with a piece of surface is labeled either symmetric or non-symmetric. We store this information compactly using a spatial octree. Although we employ a sampled representation to store symmetry information, the actual test for comparing geometry is still exact: We store the plane equations and boundary line equations of all triangles in each voxel and match them when comparing two pieces of geometry. We explicitly test for and exclude zero-area triangles where two edges are collinear up to numerical precision (which are unfortunately found frequently in real-world models). All computations of docking sites and dockers described in Section 5.1 are performed directly on the voxel grid representation. Intrinsic distances and neighborhoods are measured as graph distance in the graph of neighboring voxels,

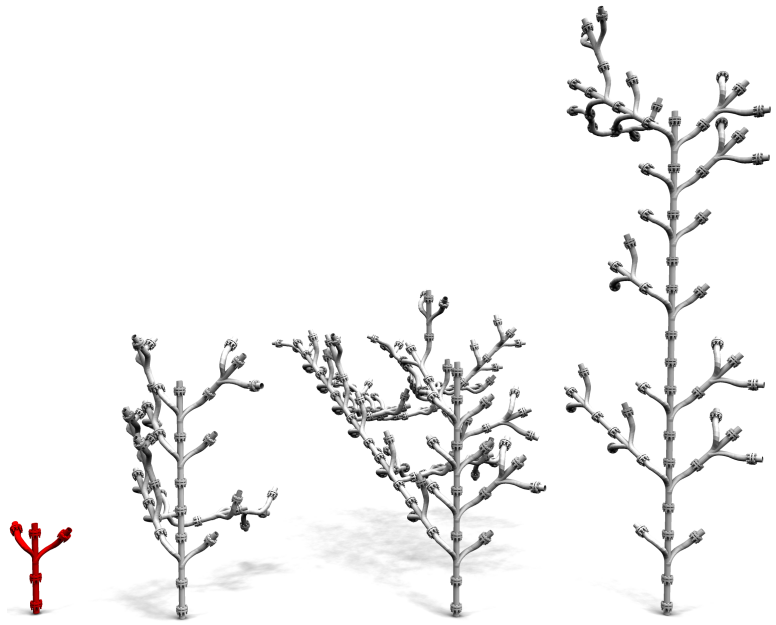


Figure 5.10: *Random variations of the "pipe tree" (original model shown in red) created by randomly applying rules from the shape grammar.*

connected by a 26-neighborhood. Once we have identified the voxels that constitute docking sites and dockers, we use the boundaries of the voxels to cut out the docking site curves and dockers out of the original triangle mesh.

Discussion: The only approximation in this strategy is in the extend of the symmetric region, which might be underestimated by at most ϵ_s. We argue that this is a reasonable strategy: First, the algorithm will converge to an exact solution with shrinking ϵ_s, and thanks to the hierarchical representation, it is no problem to use an ϵ_s much smaller than r in practice. Furthermore, the similarity parameter r is usually only a vague guess by the user so that it does not seem necessary to approximate it with very high accuracy. In our examples, we always set ϵ_s to $r/4$.

5.2.1 Symmetry Detection

Our technique requires detecting symmetries in the input model as a first step. We use a variant of the algorithm described in Chapter 4, adapted to our voxel-based representation: Whenever a rigid motion maps the input surface $\mathcal{S}$ onto itself, we have found a partial symmetry. However, not all such symmetries are useful for modeling. For example, a planar area within an object can be partially mapped onto itself by a continuous set of infinitely many transformations. However, enumerating a large number of these will not be particularly useful for modeling. We therefore limit our algorithm to find “useful” symmetries by aligning salient feature lines.

In case of scanner data, we apply the algorithm for finding feature lines as

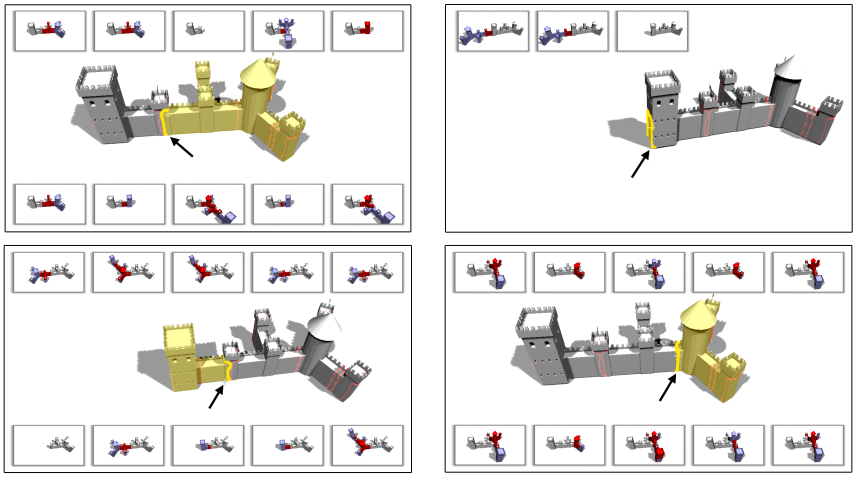
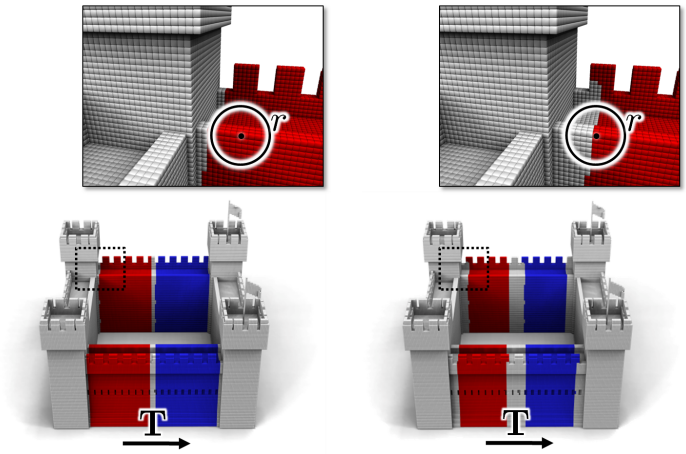


Figure 5.11: *Editing with context free grammars: possible edits for four different docking sites of the same model. The selected docking site and the docker to be replaced appear in yellow, other docking sites in light red. The rows of images above and below the model preview editing options. In these images, the red part is the new docker that will replace the yellow one. The blue parts are default geometry inserted to close the model; the user can exchange these next. Please note that many of the red dockers look similar but offer different sub-docking sites. In a context free grammar, these choices are encoded in different production rules if the alternative docking sites overlap. The accompanying video shows an interactive demonstration.*



a) discretized symmetry b) discretized r-symmetry

Figure 5.12: *Discretized r-symmetry: for a given transformation T we test every voxel for symmetry; if the content of a voxel is symmetric, we mark it accordingly (every voxel shown in red is symmetric under T and maps to the voxels shown in blue). We obtain r-symmetry by applying erosion on the symmetric voxels using a fast-marching scheme.*

described in the last chapter. For triangle meshes with perfect symmetries we can directly extract feature lines from the triangle structure: we use a triangle edge as feature if it is either a border or if the two neighboring triangles have different plane equations. Additionally, we test triangles for intersection and create a feature line at the intersection. In order to be independent of the triangulation, we merge adjacent feature lines with equivalent properties and line directions.

Next, we determine the area that is symmetric under each candidate transformation by computing the intersection of the regular voxel grid representation with a transformed version of itself. In each cell, we compare the original exact geometry, as described previously. This gives us a voxel-quantized approximation of $\xi(\mathbf{T})$. This strategy also works naturally with point cloud data since the original algorithm was designed for it. To obtain the r-symmetric set $\xi_r(\mathbf{T})$, we use a fast-marching algorithm to perform an erosion operation on the voxel grid (see Figure 5.12). In the case of raw point-cloud data from 3D scanners, we add an additional processing step that removes isolated non-symmetric voxels within symmetric voxels in order to be robust to outliers.

5.2.2 Applications

We have implemented three example modeling tools within a prototypical inverse procedural modeling application.

Creating random shape variations: It is often useful to be able to create a large number of variations of a base geometry automatically, for example for creating

background props in a game level or movie scene. We create random instance by executing random production rules from the computed shape grammar. For each rule, we check for collisions and revert to just using the original geometry of the docking sites if 10 random rules failed to work. The same strategy is also used to guarantee termination: Usually, the number of non-terminals grows faster than the likelihood of inserting terminal geometry. Therefore, we stop with original geometry after a certain number of overall replacements.

Semi-automatic modeling: We have also implemented an interactive editor to modify an existing shape according to user input (Figure 5.11). We start with the original model and display all docking sites as surface curves. The user can then "hover" over the model, docking sites are highlighted and the user can choose from the known production rules in order to change the model. Changes can be made at all levels of the hierarchy. The editor always displays a complete, r-similar model: When a new docker is inserted, its hierarchically dependent docking sites will be filled with the geometry that was originally contained in the docker, shown in a different color to indicate the default behavior.

Grid-based resizing: In addition to the context free rules used above, we also detect grid rules in the model. We let the user choose 1-grid or 2-grid rules and specify the number of repetitions.

5.3 Results

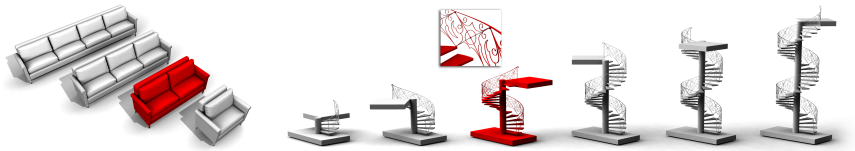
The results presented in this section are obtained from a single threaded C++ implementation of our framework running on a 2.6Ghz Core2 Duo computer with 8GB of RAM.

5.3.1 Example scenes and setup

We have chosen a number of example scenes of varying characteristics. The originals are always marked in red. Our examples include triangle meshes with exact symmetries. In addition, we use two point cloud data sets: The new town hall of Hannover (courtesy of C. Brenner, IKG Hannover) has been used previously [Pauly et al., 2008; Bokeloh et al., 2009] as benchmark data set for symmetry detection. We have manually extracted the facade and coarsely deleted some scanner artifacts. Additional noise or outlier removal turned out not to be necessary. The second is a scan of the "Zwinger" palace in Dresden, used without additional preprocessing.

5.3.2 Grid-based editing

We have applied our method to models that contain grid structures. Examples for 1-grids are shown in Figure 5.13 and 2-grids in Figure 5.14. The spiral stairs scene (b) has complex micro-geometry below the handrail, which shows the numerical robustness of our implementation. The triangle mesh of the parking structure example (Figure 5.14d) contains many of non-manifold intersections, sliver triangles



a) b)

Figure 5.13: *Two models with 1-grid structure: the sofas in a) contain a pure translational 1-grid, the spiral stairs in b) yields a helical 1-grid. Please note that the technique is robust to fine scale and highly detailed geometry of the handrail of the spiral stairs.*

and doublet triangles that cover the same area, which is typical for models that have been designed primarily for rendering. Nevertheless, we obtain stable symmetry results.

5.3.3 Manual and random modeling

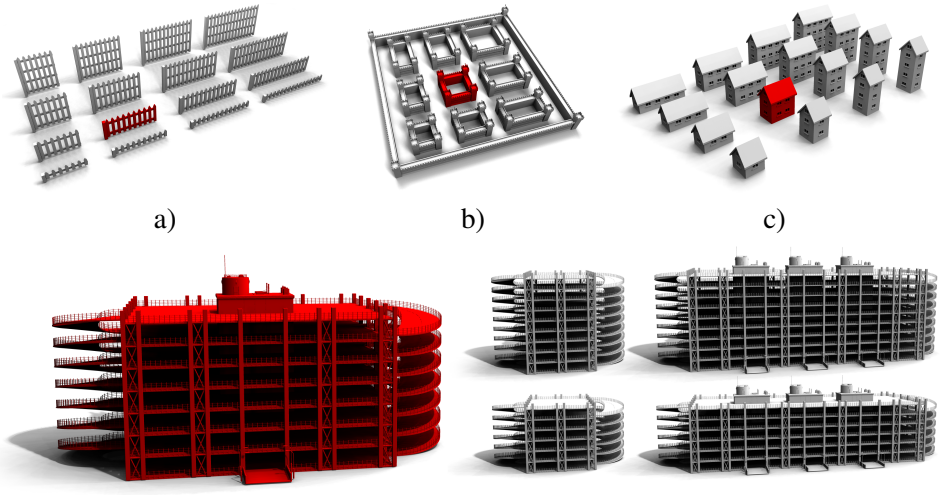
We have performed random example generation (Figure 5.10, Figure 5.15, and Figure 5.16) as well as manual modeling (Figure 5.3, Figure 5.17, and Figure 5.18). The random examples have been picked as typical examples out of a small number of random trials, all of which yielded reasonable geometry. An example grammar as computed by our algorithm is shown in Figure 5.9. The result is not as canonical as a manual, human designed grammar but simple enough to be useful in interactive manual modeling. With manual interaction, more control is possible. The editor (see Figure 5.11) is easy to use and greatly facilitates shape editing; the models shown in Figure 5.3, Figure 5.17, and Figure 5.18 were assembled in less than a minutes each.

5.3.4 Point clouds

Figure 5.16 show results for scanned facade examples. For the front of the “New Town Hall”, our algorithm has extracted 20 non-terminals corresponding to 40 dockers, all of which are of 1-grid type, due to the regular structure of the model. We use these rules to randomly insert and delete windows and the different tower elements of the facade, which lead to plausible results in all cases. Similar results are obtained for the “Zwinger” scan (courtesy of M. Wacker, HTW Dresden).

5.3.5 Analysis

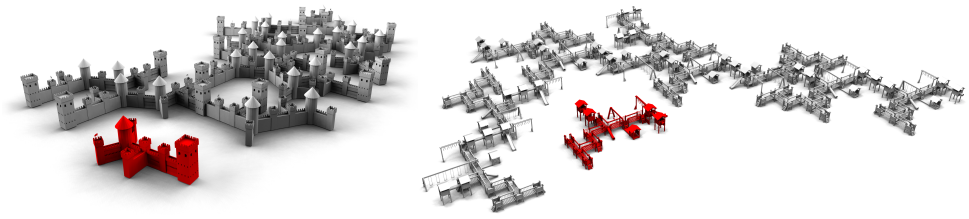
Our input scenes have a complexity ranging from 1,040 (house, Figure 5.14 c) to 500,000 (parking structure, Figure 5.14d) triangles. The new town hall facade consists of 1.2 million points. The computation times for our shape analysis are rather moderate, even for the large examples: For the facade, symmetry detection took 72 seconds, the computation of the docking sites and dockers 18 seconds,



a) b) c)

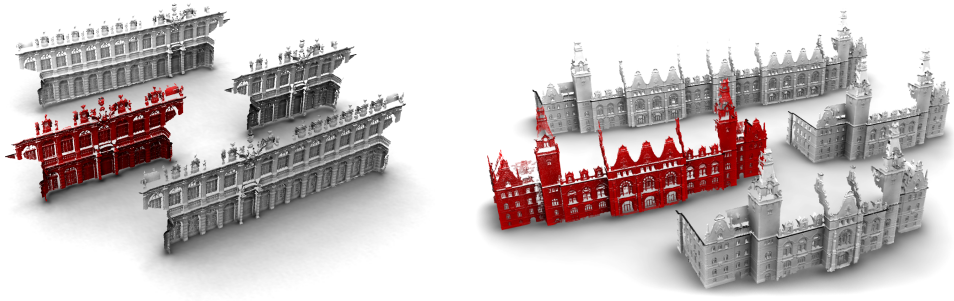
d)

Figure 5.14: *Models with 2-grid structure: The castle b) is a special (easier) case with an empty middle piece $r_{2,2}$. The parking structure d) from the Dosch Design shape collection is the most complex triangle mesh that we tested with about 500.000 triangles.*



a) b)

Figure 5.15: *Random variations of triangle meshes: both results were created by randomly applying rules from the extracted shape grammar.*



a) b)

Figure 5.16: *Random variations of point clouds: a) "Zwinger" (courtesy of M. Wacker, HTW Dresden), b) "new town hall" facade (courtesy of C. Brenner, IKG Hannover).*

and building the grammar 125 seconds. For the parking structure, the complete processing took 10 minutes, 2 minutes of which were spent on finding symmetries and extracting docking sites. This example is more time consuming because more symmetries can be found within an "exact" data set. Interactive editing and random example generation is performed in real-time, with response times (far) below one second for all examples shown in this chapter. The effect of choosing parameter r is illustrated in Figure 5.3: small values of r (middle row) produce piano keys with arbitrary grids, while a bigger value (bottom row) enforces the well-known 2/3 combinations of groups of keys. Our observation is that other than for such subtleties, the choice of r is not critical; all other examples use a fixed r that is set to 1.6% of the maximum bounding box side length of the object.

5.3.6 Limitations

Our current approach is limited to more or less exact symmetry and similarity, although some measurement noise or small inaccuracies are acceptable. However, we cannot handle any natural objects such as plants, animals, or people. Our current implementation is in addition limited to rigid similarity. With respect to shape grammars, our current applications are currently only using context free and grid-based rules, while context-sensitive modeling is still subject to future work. The rules that we extract do not cover the whole space of r-similar objects but only a subset; however, we can explicitly construct the members of this subset. In comparison to related texture-synthesis-based modeling approaches, we are not performing variational optimization. Therefore, we cannot "fit" models to arbitrary boundary conditions. In principle, the building blocks extracted by our algorithm could be used within a discrete MRF labeling algorithm to solve boundary value problems, however facing similar optimization problems as in texture synthesis. It is important to stress that texture/geometry synthesis *always* requires solving complex optimization problems, while our approach can still be used without.

Figure 5.17: *Manual editing of a bus station.*

5.4 Summary and Future Work

We have presented a theoretical framework for inverse procedural modeling of 3D objects and a practical implementation of a semi-automatic modeling system based on this framework. The main conceptual idea is to create a shape grammar that describes a large set of objects that are locally similar to a training exemplar under rigid motions. The key observation is that a grammar describing a large class of shape operations that maintain r-similarity can be directly derived from r-symmetry. The main algorithmic idea is the construction of docking sites and dockers: Whenever we can find a curve through a symmetric area that partitions the object into two pieces, we can derive a replacement operation that maintains r-similarity. A topological classification yields different types of operations (insert, delete, replace) and a hierarchical inspection of these operations then results in a context free grammar as well as grid-like replication rules. We believe that our proposal is only a first step into the mostly unexplored area of inverse procedural modeling, i.e., how to infer rules of how objects are build and structured solely from example instances. There are a number of open problems in our work that we have to leave for future work: In particular, it would be interesting to generalize our framework to other notions of similarity like affine mappings with scaling (fractal patterns) or isometric mappings (bending invariant modeling). While most of our theoretical framework covers these cases, a practical evaluation still needs to be performed. In addition to this, it is still an open question how to define a shape grammar that includes all r-similar objects; for general input models our current construction covers only a subset. In a similar direction, it would also be interesting to evaluate in how far general, non-context free shape grammars can be used for shape modeling.

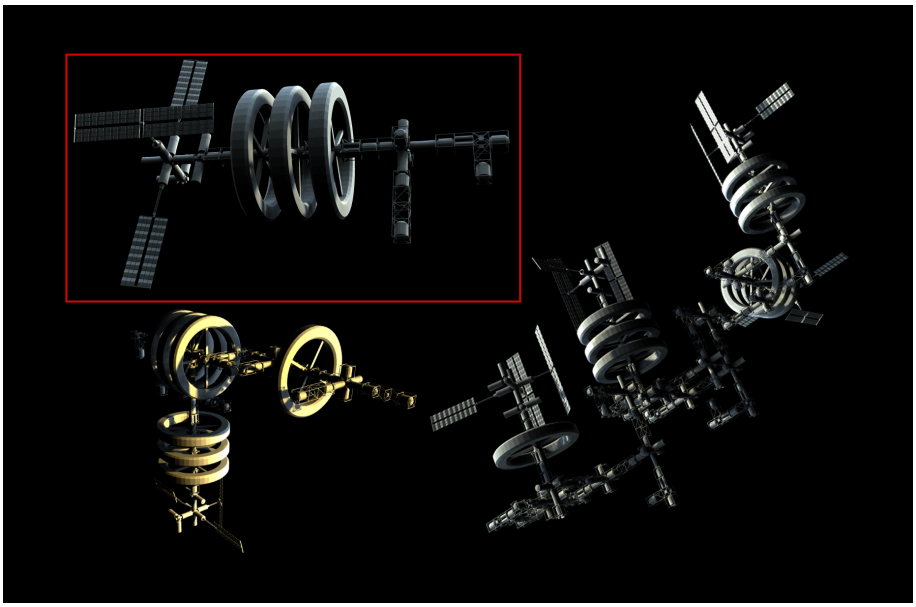


Figure 5.18: *Variations of a space station (original model shown in red frame, top-left).*

6 Pattern-Aware Shape Deformation

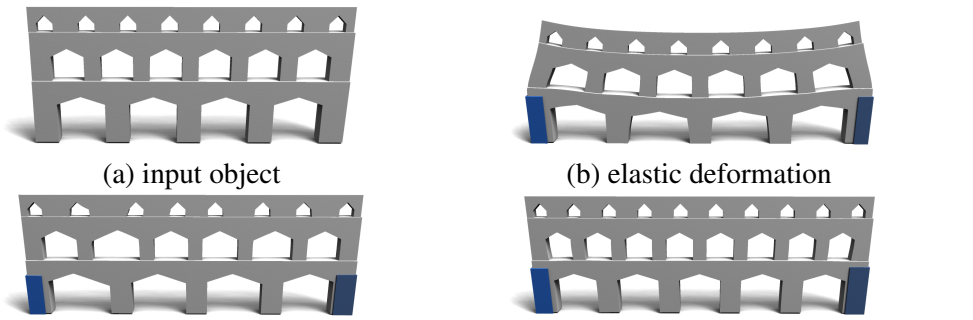
Please excuse the crudity of this model, I didn't have time to build it to scale or to paint it.

Emmett L. Brown

Building new 3D models from scratch is a tedious and time consuming task and requires artistic abilities as well as technical skills. Modification of existing 3D models presents an easier alternative to create content. Consequently, a large variety of techniques have been developed to alter shapes in such a way that they meet user defined constraints. Many techniques of this kind interpret shape deformation as a physical process and try to simulate this behavior [Terzopoulos et al., 1987; von Funck et al., 2006; Mezger et al., 2008; Botsch and Sorkine, 2008]. Other techniques opt for different properties e.g. smoothness [Sederberg and Parry, 1986; Coquillart, 1990]. Some techniques take multiple examples as input and construct a shape-space of the given class of objects [Blanz and Vetter, 1999; Allen et al., 2003; Hasler et al., 2009]. However, all these techniques ignore structural properties of a shape like symmetries or regularities, both important for the appearance of a shape.

Recent research has begun to investigate *structure-aware* shape editing tools that aim to automate the detailed manipulation required to preserve the structural relationships in a shape as it undergoes manipulation [Kraevoy et al., 2008; Gal et al., 2009; Huang et al., 2009; Wang et al., 2011; Zheng et al., 2011]. Such algorithms analyze the input shape to extract structural features and use the learned structure to assist interactive 3D modeling. They can improve the efficiency of content creation professionals and also assist inexperienced users in adapting existing content to their needs.

In this chapter, we present a structure-aware shape editing technique that detects discrete and continuous patterns in the shape and preserves these patterns under free-form deformation. A key distinguishing feature of our approach is that it can change the structure of the object by adding or removing local elements along



(a) input object (b) elastic deformation

(c) maintaining continuous patterns (d) discrete patterns, adapted repetition counts

Figure 6.1: *Overview of our approach. Given an input shape (a) and a free-form deformation applied by the user (b), our deformation model preserves continuous patterns in the shape (c) and adapts the repetition counts of discrete patterns to minimize distortion (d).*

regular patterns. This structural adaptation is integrated into a global free-form deformation framework that minimizes the overall stretch of the object.

In our approach, the user specifies a small set of constraints and the system computes a new shape that meets these constraints while preserving structural properties of the original model, as shown in Figure 1. As invariants, we extract 1-parameter groups of partial symmetries. In other words, we detect geometry that is replicated in regular patterns. This includes continuous symmetries such as straight lines as well as repeated discrete elements such as windows in a building. We formulate non-local rigidity constraints to maintain these symmetry properties in the output, and allow for adapting the number of discrete repetitions in order to reduce distortions.

In order to add and remove elements along discrete patterns with minimal distortion, we introduce *sliding dockers*. A sliding docker is an element in a local, repeated structure that interfaces with the rest of the model in such way that the structure can be independently replicated with minimal distortion. We develop an algorithm that automatically finds collections of sliding dockers in input geometry and replicates them in a way that minimizes distortions in the overall object.

6.1 Overview

Our technique is designed to preserve regular patterns in the input shape. We detect such patterns in a preprocessing step, described in Section 6.2. We then apply a continuous deformation model that tries to maintain the detected structure, as described in Section 6.3. In order to reduce distortions, we automatically insert or delete repeated elements using sliding dockers, developed in Section 6.4. In this section we give a brief overview of each component of our approach. These components are illustrated in Figure 6.1.

Input: Our technique accepts a general triangle mesh $\mathcal{S} \subseteq \mathbb{R}^3$ as input (Figure 6.1a), with no restrictions in geometry or topology (in other words, "triangle soup"). In addition, the user can select an arbitrary number of handles $\mathcal{H}_i \subseteq \mathcal{S}$ and can move and rotate them to new positions (this is an interactive process, with real-time feedback by the system). In the following, we will use $l(\mathcal{S})$ to denote the maximum side length of an axis aligned bounding box of $\mathcal{S}$; this value is used to scale relative parameters automatically.
Deformation model: The basis of our technique is a standard elastic shape deformation model [Terzopoulos et al., 1987; Sorkine and Alexa, 2007]. It computes a deformation field f that minimizes the deviation from the user's constraints and tries to keep the object as rigid as possible. In other words, the model diffuses stretch (stress tensors) as uniformly as possible across the object surface under the given constraints (Figure 6.1b). We use this behavior as a "base regularizer" with low weight, aiming at just dissipating the stretch induced by the constraints.
Shape analysis: We perform a shape analysis step in preprocessing, in order to identify regular patterns in the input geometry. We model regular patterns as one-parameter partial group structures in the symmetry structure of the object: We find parts $\mathcal{P} \subseteq \mathcal{S}$ that show up multiple times, replicated by a series of transformations $\mathbf{T}^x$, where x ranges over a continuous or integral range $\mathcal{I} \subset \mathbb{R}$, leading to continuous and discrete patterns.
Sliding dockers: For discrete patterns, we find *sliding dockers*, which are building blocks that can be replicated when the object is locally stretched. Sliding dockers are cut out of the input surface in a way that the boundaries fit seamlessly when multiple pieces are attached to each other regularly. In addition, the boundaries of this repeated region are slippable, such that any repetition count yields closed geometry.
Continuous, structure-aware deformation: Using the structural knowledge gained in preprocessing, we add constraints to our deformation model that aim at preserving the patterns in the shape. These constraints are given a higher weight than the elastic regularizer, thus dominating the deformation results (Figure 6.1c).
Discrete relaxation: We measure the stretch in the continuously deformed model and automatically insert or delete sliding dockers to relax the stretch in the model. Such automatic structure adaptation allows a broader range of deformations to be applied without violating the natural appearance of the object (Figure 6.1d).
The following sections describe each of these components in turn.

6.2 Pattern-Based Structure Model

Our approach begins with a preprocessing phase, which analyses the input geometry to detect structural regularity in the form of partial regular patterns. These patterns will be kept invariant in the later editing process. Regular patterns are defined with respect to a *group of admissible transformations*:
Transformations: Let $\mathcal{G}$ be a group of bijective, continuous mappings $\mathbb{R}^3 \to \mathbb{R}^3$.

We will restrict our consideration to rigid motions (rotations and translations).

Replications: For a rigid motion $\mathbf{T} \in \mathcal{G}$, where $\mathbf{T}$ is not a rotation by $180°$, we can define continuous powers of $\mathbf{T}$: for $x \in \mathbb{R}$ let $\mathbf{T}^x := \exp(x \log \mathbf{T})$. Then, for a set $A \subseteq \mathbb{R}$, we will use the the notation $\mathbf{T}^A := \{\mathbf{T}^x | x \in A\}$ in order to denote the set of powers of $\mathbf{T}$. Furthermore, for $\mathcal{P} \subset \mathbb{R}^3$, we write $\mathbf{T}^A(\mathcal{P}) := \cup_{x \in A} \mathbf{T}^x(\mathcal{P})$ to denote replications of $\mathcal{P}$. In particular, $\mathbf{T}^{\mathbb{R}}(\mathcal{P})$ denotes the *continuous kinematic surface* that replicates $\mathcal{P}$ continuously.

6.2.1 Partial Regular Patterns

Our goal is to preserve the symmetry structure of the input $\mathcal{S}$ under deformations while admitting insertions and deletions of parts. However, this intuitive notion is not easy to formalize. A first attempt would be to look at the global symmetry groups of $\mathcal{S}$:

Symmetry groups: The set of all operations $\mathbf{T} \in \mathcal{G}$ that maps $\mathcal{S}$ to itself, i.e., $\mathbf{T}(\mathcal{S}) = \mathcal{S}$ forms a subgroup of $\mathcal{G}$. It characterizes the global symmetries of $\mathcal{S}$.

There are now two major obstacles: First, the structure of the symmetry group can be involved. Because of potentially non-commutative transformations, it is in the general case difficult to find a canonical representation [Miller, 1972]. The second problem is that the restriction to global symmetries is not satisfactory: Many objects have partial symmetries that do not form global symmetry groups but are still perceptually important (such as a row of windows in a building). We therefore restrict ourselves to a simple, tractable model: We only look at *partial 1-parameter groups* (for short: *patterns*) and characterize the overall shape by their interaction. These structures only capture commutative 1-parameter subgroups of more complex symmetry groups and, in addition, permit partial coverage:

Partial 1-parameter symmetry groups ("patterns"): Consider $\mathcal{P} \subseteq \mathcal{S}$ and $\mathbf{T} \in \mathcal{G}$, and let $\mathcal{I} \subset \mathbb{R}$ be a real interval. If we have $\mathbf{T}^{\mathcal{I}}(\mathcal{P}) \subseteq \mathcal{S}$, we have found a *continuous partial 1-parameter symmetry group* of $\mathcal{S}$. If $\mathcal{I} \subseteq \mathbb{Z}$ is an integer interval with at least three elements, we have found a *discrete* partial 1-parameter symmetry group. For brevity, we will call these structures continuous and discrete *(regular) patterns*, respectively.

Normalization: The definition above usually gives rise to an infinite number of patterns. We therefore normalize the representation. For discrete symmetries, we choose maximal sets $\mathcal{P}$ that yield the same transformation set. In the continuous case, moving a piece $\mathcal{P}$ along a continuous rigid motion $\mathbf{T}^x$ yields a *kinematic surface* [Gelfand and Guibas, 2004]. We can describe the pattern by just the maximal surface $\mathcal{P}$ that is slippable under a constant instantaneous velocity $\log \mathbf{T}$. Using this representation, the interval $\mathcal{I}$ vanishes and we will omit it in the following when denoting continuous symmetries.

Regular Patterns of $\mathcal{S}$: Using these conventions, we obtain a finite set of discrete regular patterns $R_D = \{(\mathcal{P}_1, \mathbf{T}_1, \mathcal{I}_1), ..., (\mathcal{P}_N, \mathbf{T}_N, \mathcal{I}_N)\}$ and another finite set of continuous regular patterns $R_C = \{(\mathcal{P}_1, \mathbf{T}_1), ..., (\mathcal{P}_M, \mathbf{T}_M)\}$.

We now discuss how to detect these structures automatically in an input triangle mesh.

6.2.2 Computational Framework

Discrete Patterns: We compute the discrete patterns by a symmetry analysis similar to to our techniques in Chapter 4 and Chapter 5: We detect sharp creases in the input mesh and combine pairs of adjacent, non-collinear creases to form "bases" . Two base pairs are potentially corresponding if they have matching length and enclose the same angle. Very small feature lines (below 2.5% $l(\mathcal{S})$) are removed for efficiency reasons. We now use a RanSaC procedure to compute regular patterns: Random pairs of potentially corresponding bases are chosen and the relative transformation $\mathbf{T}$ is computed. We search for all potentially corresponding bases that are located at positions $\mathbf{T}^x$ for some $x \in \mathbb{R}$. This gives us initial pattern candidates. The next step is to extract generator transformations $\mathbf{T}$ that generate the 1-parameter groups $\mathbf{T}^i, i \in \mathcal{I} \subset \mathbb{Z}$: We look at all pairwise transformations between candidate bases. For each pair, we compute the number of bases that lie on the grid $\mathbf{T}^{\mathbb{Z}}$ induced by the two bases. We output the choice of generator that yields the largest integer interval $\mathcal{I}$ of matching bases and exclude these from further processing. We iterate until no more valid patterns are found.

Continuous patterns: Continuous symmetries with respect to rigid motions are detected by slippage analysis, following the algorithm of [Gelfand and Guibas, 2004]. For a point $\mathbf{x} \in \mathcal{S}$, the algorithm considers a small local neighborhood $N(\mathbf{x})$ and determines how the distance between $\mathbf{T}(N(\mathbf{x}))$ and $\mathcal{S}$ changes for infinitesimally small motions $\mathbf{T} \in \mathcal{G}$. Technically, we have to consider the Hessian of the surface distance function under rigid motions. An eigenanalysis then yields (near-) zero eigenvalues for *slippable motions*. These are the transformations of continuous symmetries. There are six different types of slippable surfaces, with 1-3 degrees of freedom (see Figure 2.1). Parts $\mathcal{P}$ that have the same continuous symmetry properties are extracted by simple region growing (see [Gelfand and Guibas, 2004] for implementation details).

6.3 Deformation Model

In this section, we describe the global continuous deformation model that serves as the basis for our deformation framework. First, we describe the representation of the deformation function (Section 6.3.1). Second, we review the standard elastic deformation model, which serves as our base regularizer (Section 6.3.2). Third, we introduce additional structure-aware constraints in order to preserve regular patterns (Section 6.3.3).

6.3.1 Representation

In order to compute a deformation, we embed the surface $\mathcal{S}$ into a volume $\mathcal{V} \subset \mathbb{R}^3$, $\mathcal{S} \subset \mathcal{V}$, and deform this volume using a deformation field $f : \mathcal{V} \to \mathbb{R}^3$. This approach has the benefit of making the deformation independent of the representation of $\mathcal{S}$ so that arbitrary types of input geometry and general surface topology can be handled easily. Following [Huang et al., 2006a; Sumner et al., 2007], we use a subspace method to discretize f, i.e., we use a low-dimensional basis for representing the deformation: We create a number of nodes $\mathbf{x}_1, ..., \mathbf{x}_k \subset \mathbb{R}^3$ and center radial basis functions b_i around them to define the deformation field:

$$f(\mathbf{x}) = \sum_{i=1}^{K} \mathbf{u}_i b_i(\mathbf{x}) \tag{6.1}$$

Here, $\mathbf{u}_i \in \mathbb{R}^3$ are the deformed target positions of the nodes $\mathbf{x}_i$. As basis functions, we employ moving-least-squares (MLS) meshless basis functions of linear precision, based on a finite support Wendland kernel, as proposed in [Adams et al., 2008]. These functions are able to represent smooth deformation fields with a small number of nodes. In particular, affine motions such as rotations are interpolated exactly. We place the nodes by discretizing $\mathcal{V}$ to a regular grid of user specified spacing h (Figure 6.2). We set the support of the basis function to $2h$ to make sure that at least two basis functions overlap each surface point in x-, y-, and z-direction. The volume $\mathcal{V}$ itself is created by offsetting $\mathcal{S}$ by h in all directions (i.e., a Minkowski sum of a sphere of radius h and $\mathcal{S}$). This guarantees that the basis functions and their derivatives are well defined on $\mathcal{S}$.

Remark: In the following, we use two basic numerical discretization constants. The first, h, determines the resolution of the deformation field, which is typically in the range of $5\%\ l(\mathcal{S})$. In addition, we also use smaller constant ϵ (in the range of $1\%\ l(\mathcal{S})$) for discretizing other functions, such as symmetry information and to form neighborhoods for slippage analysis.

f is determined by a variational approach: We set up an energy function $E(f)$ that is minimized by an optimal f:

$$E = E_c + \lambda_r E_r + \lambda_c E_c + \lambda_d E_d \tag{6.2}$$

E_c describes user constraints and E_r is the base-regularizer that creates elastic behavior. These two terms correspond to a standard elastic shape deformation approach. We then add two additional terms to preserve the pattern structure: E_c preserves continuous patterns such as straight lines, and E_d preserves discrete patterns. The λ_* control the influence of the different regularizers relative to the user constraints. We typically use $\lambda_r = 0.01$ and $\lambda_c = \lambda_d = 1$.

6.3.2 Elastic Deformation

The first energy term E_c accounts for user constraints. We use the standard "handle" model [Bendels et al., 2003; Botsch and Kobbelt, 2004] where parts $\mathcal{H}_i \subseteq \mathcal{S}$

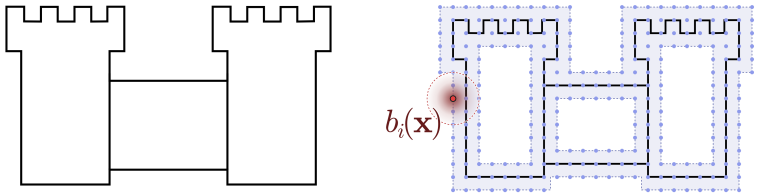


Figure 6.2: *We discretize the deformation field f using meshless basis functions on a volumetric grid.*

of the input surface can be translated and rotated in space:

$$E_c(f) = \sum_{\mathcal{H}_i \in \mathcal{H}} \int_{\mathcal{H}_i} \left(f(\mathbf{x}) - (\mathbf{R}_i^{(\mathcal{H})}\mathbf{x} + \mathbf{t}_i^{(\mathcal{H})}) \right)^2 d\mathbf{x} \tag{6.3}$$

The second term E_e is the elastic deformation energy. We employ a standard formulation based on a Poisson system [Sorkine et al., 2004] with co-rotated local frames [Müller et al., 2002; Sorkine and Alexa, 2007], adapted to the volumetric settings [Zhou et al., 2005]: We connect all pairs of nodes with overlapping shape functions and preserve their distance vectors:

$$E_r(f) = \sum_{i=1}^{K} \sum_{j \in N(i)} \omega_{i,j} \left(\mathbf{u}_i - \mathbf{u}_j - \frac{1}{2}(\mathbf{R}_i + \mathbf{R}_j)\,(\mathbf{x}_i - \mathbf{x}_j) \right)^2 \tag{6.4}$$

Here, $N(i)$ denotes accordingly the set of indices of nodes adjacent to node $\mathbf{x}_i$. The $\omega_{i,j}$ are the weights of their coupling, which we set according to the Wendland kernel of the MLS basis (see [Zhou et al., 2005] for a more sophisticated scheme). The variables $\mathbf{R}_i$ are rotation matrices at each node $\mathbf{x}_i$ that are optimized along with the node displacements.

Numerical solution: In order to solve for a minimum of the energy, we determine the derivative with respect to the $\mathbf{u}_i$, which is a linear, Laplacian-type system, and set it to zero. Next, we update the rotation $\mathcal{R}_i$ by estimation from their neighbors. This procedure is iterated until convergence. Details can be found in [Sorkine and Alexa, 2007]. Because of the special structure of this system, only the right-hand side changes during the iterations. Therefore, it is possible to prefactor the matrix so that the inversion can be solved by sparse matrix-vector products, leading to a substantial speed-up. As suggested in [Sorkine and Alexa, 2007], we employ the TAUCS library for sparse Cholesky factorization [Toledo, 2003].

6.3.3 Structure Aware Deformation

We now augment our deformation model so that it better preserves the structure of the deformed geometry. We first employ a general anisotropic deformation model in order to favor a local preservation of pattern structures. Secondly, we add global constraints that preserve continuous and discrete patterns.

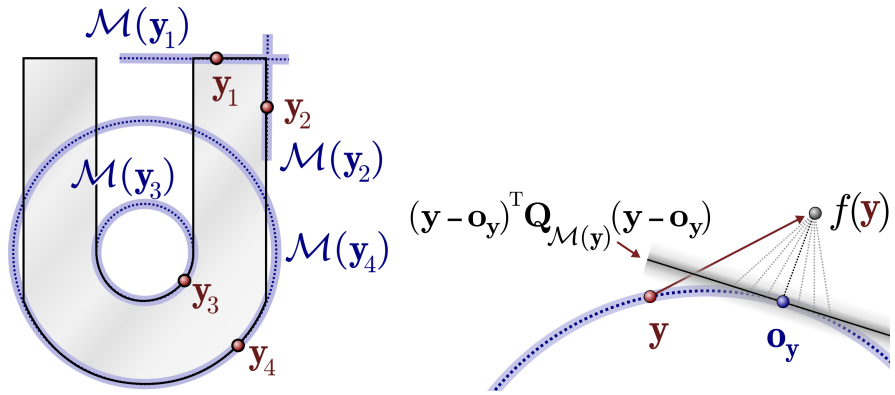


Figure 6.3: *Constraint manifolds are constructed to preserve continuous pattern structures.*

Local Constraints

Locally, we would like geometry to deform in a way that preserves continuous symmetries. If we look at this from the local perspective of the elastic regularizer, this means that we would like the deformation to happen along slippable motions rather than orthogonal to them, because this will only change the parametrization, but not the geometric shape. Accordingly, we augment Equation 6.4 by using an anisotropic error quadric in order to weight deformations. We replace the isotropic error term $(\mathbf{u}_i - \mathbf{u}_j - \frac{1}{2}(\mathbf{R}_i + \mathbf{R}_j)\,(\mathbf{x}_i - \mathbf{x}_j))^2 =: (\mathbf{d}_{i,j}^{el})^2$ by:

$$\left(\mathbf{d}_{i,j}^{el}\right)^\top \frac{1}{2}\left(\mathbf{Q}_s(\mathbf{x}_i) + \mathbf{Q}_s(\mathbf{x}_j)\right)\left(\mathbf{d}_{i,j}^{el}\right) \tag{6.5}$$

where $\mathbf{Q}_s(\mathbf{x})$ is computed by a translational slippage analysis:

$$\mathbf{Q}_s(\mathbf{x}) = \int_{N_h(\mathbf{x})} \mathbf{n}(\mathbf{y}) \cdot \mathbf{n}(\mathbf{y})^\top d\mathbf{y} + 0.01 \cdot \mathbf{I}. \tag{6.6}$$

Here, $\mathbf{n}(\mathbf{x})$ is a unit surface normal at $\mathbf{x} \in \mathcal{S}$, and $N_h(\mathbf{x})$ is the Euclidean h-neighborhood of $\mathbf{x}$ in $\mathcal{S}$. Intuitively, this can be explained as an average of planar constraints: At each point, the outer products create quadrics that penalize deviations in normal direction only; tangential motions have zero cost. For complex geometry, the costs in different directions are averaged. Thus, a straight line will penalize anything but motions in its tangential direction and irregular geometry will resist any deformation. As we still need a base regularizer that diffuses stretch, even along perfectly straight lines, we add 1% of the identity matrix.

Global Constraints

Our local constraints consider only translational patterns. In addition, the effect of the local model weakens with distance: Extended objects such as straight lines or flat planes can still show significant global bending. Increasing the weights could

in principal solve this problem but would lead to an impractically ill-conditioned numerical system. Therefore, we introduce explicit global constraints to maintain general patterns globally.

We address the continuous patterns first. The discrete case is discussed afterwards and requires only a few minor modifications. Continuous symmetries are obtained by slippage analysis [Gelfand and Guibas, 2004]. Let $\mathcal{P}$ be a part of constant slippage. It can have up to three slippable motions that are described by d matrices $\mathbf{T}_1, ..., \mathbf{T}_d$. For each point $\mathbf{y} \in \mathcal{P}$ we then consider the kinematic curve or surface

$$\mathcal{M}(\mathbf{y}) = \mathbf{T}_1^{\mathbb{R}} \cdots \mathbf{T}_d^{\mathbb{R}}(\mathbf{y}), \tag{6.7}$$

which is the *constraint manifold* for point $\mathbf{y}$ (see Figure 6.3, left). We describe the tangent space of $M(\mathbf{y})$ by a local quadratic energy: In the surface case, let $\mathbf{t}_1(\mathbf{y}), \mathbf{t}_2(\mathbf{y})$ denote two orthonormal tangent vectors of $\mathcal{M}(\mathbf{y})$. We then form a local error quadric by

$$\mathbf{Q}_{\mathcal{M}}(\mathbf{y}) = \mathbf{I} - \mathbf{t}_1(\mathbf{y})\mathbf{t}_1(\mathbf{y})^\top + \mathbf{t}_2(\mathbf{y})\mathbf{t}_2(\mathbf{y})^\top. \tag{6.8}$$

It has zero eigenvalue in tangential directions and one in the orthogonal direction. For the 1-slippable case, the same construction is made with one vector, and a single zero eigenvector. Given k slippable parts $\mathcal{P}_1, ..., \mathcal{P}_k$ and corresponding sets of motions, we then build the following global constraint energy that preserves continuous symmetries:

$$E_c \sim \sum_{i=1}^{k} \int_{\mathcal{P}_i} \mathrm{dist}(\mathbf{y}, \mathcal{M}(\mathbf{y}))^2 d\mathbf{y} \tag{6.9}$$

We measure the squared distance only in direction orthogonal to the tangent space of the constraint manifold, i.e., the distance is computed as $(\mathbf{y} - \mathbf{o}_\mathbf{y})^\top \mathbf{Q}_{\mathcal{M}}(\mathbf{y})(\mathbf{y} - \mathbf{o}_\mathbf{y})$, where $\mathbf{o}_\mathbf{y}$ is the closest point to $\mathbf{y}$ in $\mathcal{M}(\mathbf{y})$. Figure 6.3 (right) illustrates the scheme.

The constraint manifolds $\mathcal{M}(\mathbf{y})$ are not fixed during editing. Maintaining exactly the same constraint manifolds as the original input would prohibit many desirable changes to the shape. Therefore, we dynamically recompute them. We have chosen to make the translation flexible and keep the rotation fixed for continuous patterns. This leads to a behavior that is often more intuitive to the user, as dominant pattern directions retain their orientation during deformation. Translation invariance is obtained by expressing the constraints in terms of difference vectors in the actual numerical formulation, described below.

Numerical implementation: We identify the region $\mathcal{P}_i$ of each slippable part and sample them uniformly with points $\mathbf{q}_j^{(i)}, j = 1...n_i$ of spacing h using Poisson disc sampling. We then connect the points with their centroid $\mathbf{c}^{(i)}$ and form distance vectors between the centroid and all other sample points, which yields a star geometry (see Figure 6.4). The original, constant distance vectors are $\mathbf{d}_j^{(i)} = \mathbf{q}_j^{(i)} - \mathbf{c}^{(i)}$.

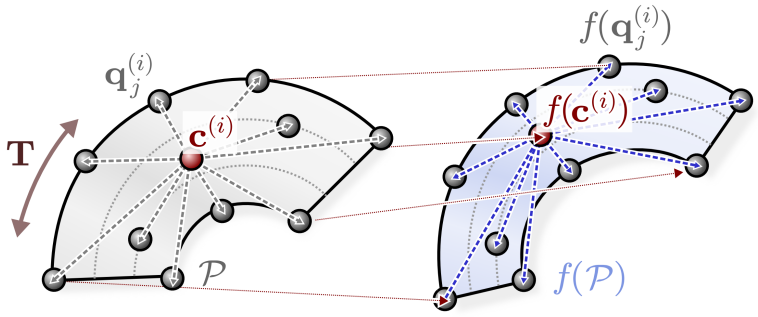


Figure 6.4: *Global constraints are implemented as Poisson constraints (maintaining difference vectors) on a star geometry. By changing the target vectors, the constraints can be updated dynamically without changing the matrix factorization.*

The distance vectors in the deformed model are given by $f(\mathbf{d}_j^{(i)}) := f(\mathbf{q}_j^{(i)}) - f(\mathbf{c}^{(i)})$. We then minimize the differences in a least squares sense:

$$E_c \quad = \quad \sum_{i=1}^{k} \sum_{j=1}^{n_i} \left[f(\mathbf{d}_j^{(i)}) - \mathbf{d}_j^{(i)} \right]^\top \mathbf{Q}_{\mathcal{M}}(\mathbf{q}_j^{(i)}) \left[f(\mathbf{d}_j^{(i)}) - \mathbf{d}_j^{(i)} \right] \quad (6.10)$$

Weighting by the error quadric $\mathbf{Q}_{\mathcal{M}}(\mathbf{q}_j^{(i)})$ constrains the deviation to the tangent space of the constraint manifolds. Again, only constants in Equation 6.10 change, so that only the right-hand side of the linear system needs to be updated. This permits prefactorization, which is crucial for achieving real-time performance.

Discrete patterns: In the discrete case, we use almost exactly the same constraints. We obtain 1-dimensional constraint manifolds as $\mathcal{M}(\mathbf{y}) = \mathbf{T}^{\mathbb{R}}(\mathbf{y})$, where $\mathbf{T}$ is the constant, finite transformation that links two elements in the discrete pattern. In addition to that, we have to make one important difference in the discrete case: For continuous patterns, moving surface points along their constraint manifold usually does not change the geometry substantially. In the discrete case, however, tangential drift is clearly noticeable because we have complex, non-slippable geometry that is replicated in discrete entities. We therefore modify the constraints to enforce a constant step size: We use quadrics $\mathbf{Q}_{\mathcal{M}}(\mathbf{y}) = \mathbf{I}$ of full rank and use an equidistant stepping $\mathbf{T}^{\mathbb{Z}}(\mathcal{P})$ to constraint difference vectors between corresponding parts.

In order to retain full flexibility of the edited shape, we recompute $\mathbf{T}$ in each step by least-squares fitting to the deformed geometry. For this, we fit an affine map to all correspondence vectors simultaneously that connect any pair of neighboring pattern elements $f(\mathbf{T}^i(\mathcal{P})), f(\mathbf{T}^{i+1}(\mathcal{P}))$ and project to rigid motions by a polar decomposition.

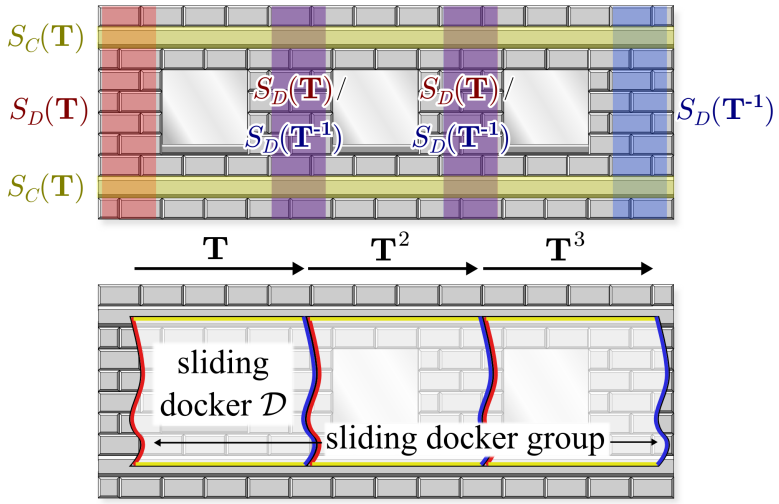


Figure 6.5: *A sliding docker has two different kinds of boundaries: A boundary orthogonal to the motion direction that cuts through a symmetric area (red) and a sliding boundary within motion direction that cuts through continuously symmetric area (yellow). Because of its symmetric properties, the transformed discrete boundary is entirely located in a region that is symmetric under the inverse transformation (blue).*

6.4 Sliding Dockers

In this section, we describe how our framework adapts the repetition count of discrete patterns in order to reduce stretch. In Sections 6.4.1 and 6.4.2 we examine the discrete patterns more closely and try to decompose their geometry into *sliding dockers* that allow changing the repetition count seamlessly. In Section 6.4.3 we describe how sliding dockers are integrated into the deformation framework.

The interactive deformation proceeds in two steps. First, we let the user deform the object. In areas covered by discrete patterns, the anisotropic deformation weights (Equation 6.6) are changed such that deformation along motion field $\partial_x \mathbf{T}^x$ incurs minimal penalties. In this step, the pattern area acts as a *placeholder*, allocating space for sliding dockers along the pattern's motion direction. In the second step, we compute the stretch within the placeholder, round it, and insert an adapted number of instances. Then the deformation is recomputed for the new composition of the object. The two deformation steps are always performed in sequence and only the adapted shape is presented to the user.

6.4.1 Defining Sliding Dockers

Our first task is to identify pieces of geometry that can be replicated. Let $\mathbf{T}^{\mathcal{I}}(\mathcal{P}) \subset \mathcal{S}$ be a discrete pattern, as computed in Subsection 6.2.2. We now need to determine whether it contains elements that can be replicated while continuously interfacing

with each other and with existing geometry.

As shown in Figure 6.5, such elements have to meet two types of boundary conditions. First, boundaries orthogonal to the motion field have to match each other; we therefore require *symmetry* of this geometry with respect to $\mathbf{T}$. Second, in direction tangential to the motion, we require *slippability* with respect to $\mathbf{T}$; by changing the repetition count, the boundaries of the pattern and the rest of the geometry will slide with respect to each other, and slippability will ensure that we always have matching geometry.

We perform symmetry analysis to find all geometry within $\mathcal{S}$ that is symmetric with respect to $\mathbf{T}$. We denote this geometry by $S_D(\mathbf{T})$:

$$S_D(\mathbf{T}) := \{\mathbf{x} \in \mathcal{S} | \mathbf{T}(\mathbf{x}) \in \mathcal{S}\} \tag{6.11}$$

It is easy to see that the image of $S_D(\mathbf{T})$ under $\mathbf{T}$ is $S_D(\mathbf{T}^{-1})$; in other words, this is the area the symmetry transform maps to. By slippage analysis, we obtain the subset of $\mathcal{S}$ that is slippable with respect to $\mathbf{T}$. We denote this set by $S_C(\mathbf{T})$.

Consider a piece of geometry $\mathcal{D} \subset \mathcal{S}$. We say that $\mathcal{D}$ is a *sliding docker* with respect to $\mathbf{T}$ if the following two conditions hold. First, the boundary $\partial\mathcal{D}$ must be located entirely in either $S_D(\mathbf{T})$, $S_D(\mathbf{T}^{-1})$, or $S_C(\mathbf{T})$. Second, for every point $\mathbf{x} \in \partial\mathcal{D}$ in $S_D(\mathbf{T})$, the corresponding point $\mathbf{T}(\mathbf{x}) \in S_D(\mathbf{T}^{-1})$ must be included in the boundary $\partial\mathcal{D}$, and vice versa. In other words, we cut out a sliding docker by cutting through symmetric geometry and slippable area along the motion of the pattern; when cutting through the symmetric area, we need to use matching cut lines within $S_D(\mathbf{T})$ and $S_D(\mathbf{T}^{-1})$ so that the pieces fit together seamlessly later (see Figure 6.5).

We can easily extend this definition to a whole array of sliding dockers. In order to find n matching dockers simultaneously, we require that the two boundary conditions are met by n replicated pieces along the motion direction, namely $\{\mathcal{D}, \mathbf{T}(\mathcal{D}), \mathbf{T}^2(\mathcal{D}), ..., \mathbf{T}^{n-1}(\mathcal{D})\}$. We call such an ensemble a *sliding docker group*.

6.4.2 Finding Sliding Dockers

In order to find sliding dockers, we first need to compute the symmetry information. We use the same computational framework as in the previous chapter: Transformation candidates are estimated by matching feature lines, and we obtain the slippable motions from slippage analysis (Section 6.2.2).

Motion space transform: In order to simplify further computations, we perform a transformation into *motion space* (Figure 6.6). In this space, one axis corresponds to the motion $\mathbf{T}^x$, while the other two axes y, z are Euclidean. In order to set up the motion space transform, we first determine whether $\mathbf{T}$ is a rotation or a pure translation. For a pure translation, we obtain motion coordinates by building a new coordinate frame where the first axis is pointing in the direction of the translation. For rotations, we compute the rotation axis, which gives us a transformation into cylindrical coordinates around this axis. (We omit the case of helical motions for

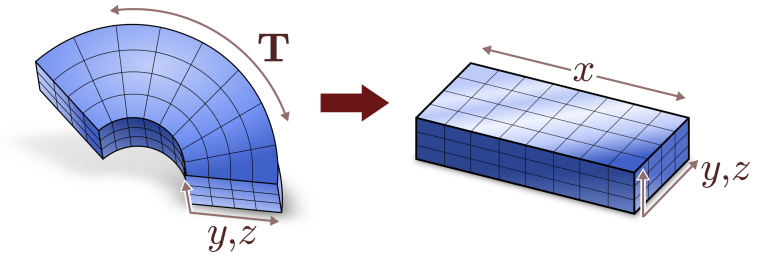


Figure 6.6: *Transformation into motion space. Similar to a decomposition into polar coordinates, we represent points by a 1D motion coordinate along* $\mathbf{T}$ *and 2D coordinates orthogonal to it.*

simplicity.) As notational convention, we will denote the motion coordinate as the x-axis of the motion space.

Next, we build a voxel grid in motion space to represent the symmetry information (see Figure 6.7 for an illustration). The side length in the motion dimension and Euclidean dimensions is chosen such that the spacing in world coordinates is not larger than the discretization constant ϵ. Furthermore, we denote by $l_{\mathbf{T}}$ the (integer) number of voxels that represent one application of the motion $\mathbf{T}$.

For each discrete regular pattern $(\mathcal{P}, \mathbf{T}, \mathcal{I}) \in R_D$, we transform the scene into the motion space of $\mathbf{T}$ (this is sped up by collecting all patterns that have the same motion space). We now retrieve the geometry in every non-empty voxel (i, j, k) and match the content against voxels $(i + l_{\mathbf{T}}, j, k)$, corresponding to the transformed geometry. Matching voxels are tagged as symmetric. Next, we compute the slippability of each voxel and check for each $\mathbf{T}$-slippable voxel (i, j, k) whether all voxels $(i, j, k), (i + 1, j, k), ..., (i + l_{\mathbf{T}}, j, k)$ are $\mathbf{T}$-slippable as well. If so, we mark the voxel as *fully slippable*. This means that the geometry at this voxel v, as well as all geometry along the motion $\mathbf{T}^{[0,1]}(v)$ is slippable, which is what we need to cut out a sliding docker. We perform this analysis for all non-empty voxels, as well as for empty voxels that are direct neighbors of occupied ones. Empty voxels must map to other empty voxels in order to be fully slippable.

Extracting sliding dockers: After this precomputation, finding a sliding docker is simple. We start at a symmetric voxel and grow in the (y, z)-plane of the motion space until we either hit a non-symmetric voxel, or a fully slippable voxel. If we hit a single non-symmetric voxel, we dismiss the whole attempt. If we only end at fully slippable voxels (including the empty fully slippable ones), we have found a sliding docker: We can just cut out an extrusion of the visited region in x-direction in motion space. By transforming back into world coordinates, we obtain the final sliding docker. When performing this computation, we always try to find a maximal sliding docker group by checking for symmetry and full slippability with respect to $\mathbf{T}, \mathbf{T}^2, ...$ simultaneously (this corresponds to testing voxels in multiples of $l_{\mathbf{T}}$ apart in the x-direction).

The whole computation is attempted for each base of a detected pattern. This

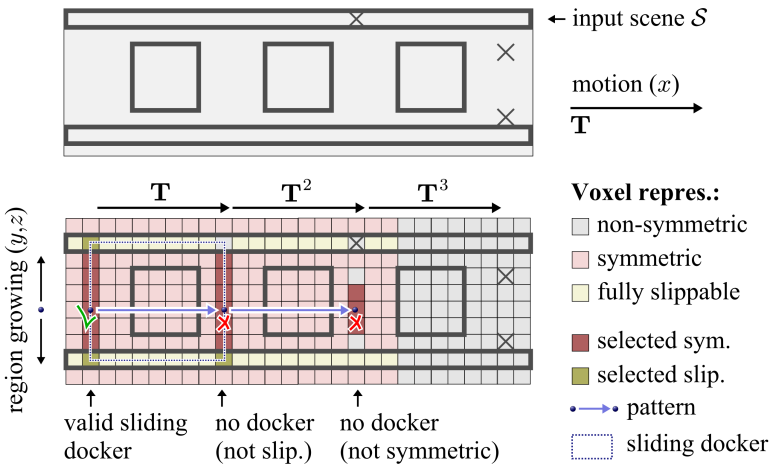


Figure 6.7: *Sliding dockers are extracted by region growing in motion space. Starting from a pattern base, we grow orthogonal to the motion direction, using only symmetric voxels. We proceed until either hitting only fully slippable voxels (success) or an unsymmetric voxel (failure).*

yields a large number of sliding dockers, most of which overlap. In order to remove overlapping pieces, we use a simple greedy algorithm: We take the largest sliding docker group (i.e., the one with the highest repetition count) and delete all overlapping sliding docker groups. This is iterated until no more sliding docker groups are found.

6.4.3 Using Sliding Dockers

We can now integrate the sliding dockers into our deformation framework. First, we have to set up the first of the two deformation steps. We mark all areas that are covered by a sliding docker group. At each such point, we deactivate all regularizers except from the elastic deformation energy. Let $\mathbf{t}(\mathbf{y})$ be a normalized vector parallel to the tangent $\partial_x \mathbf{T}^x(\mathbf{y})$ of the motion field. We then set the error quadric of the elastic deformation model (Equations 6.5,6.6) to $\mathbf{I} - \mathbf{t}\mathbf{t}^\top + 0.01 \cdot \mathbf{I}$. This makes the geometry easily stretchable in the pattern direction. For the rest of the model, we use all energy terms as previously described, including global and local pattern preserving constraints.

We then solve the resulting system. In the result, we measure the stretch of the pattern region by integrating along lines of the motion direction: We connect corresponding points in neighboring instances of the pattern elements and compute the average length. Dividing the value for the deformed and undeformed state gives us a *stretch factor* F. We multiply this factor by the number of original repetitions and round it to the nearest integer to determine the number of elements to insert.

For inserting elements, we again use the motion space. We scale the elements by the inverse stretch factor in the x-direction of the motion space, concatenate

the pieces, and backtransform into world coordinates. We then replace the original pattern with the adapted one.

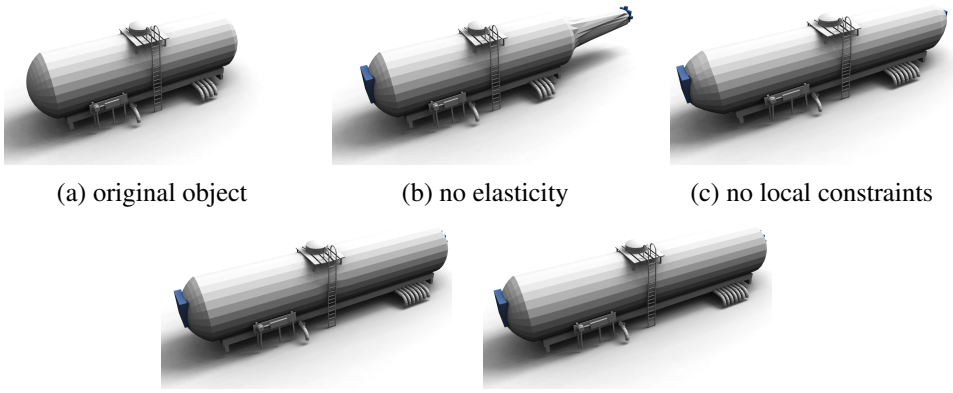
Next, we need to make sure that the elastic deformation model undoes the stretch: If we add more elements, this means that we squeeze smaller replicas into the original space. The energy of Equation 6.4 would then try to preserve this configuration in an as-rigid-as-possible manner. Therefore, we augment the distance vectors: Instead of the distances of the squeezed elements, we employ the original distance vectors. For basis functions that overlap regions that are stretched by different factors, we compute a weighted average according to the respective kernel function of that node.

As error quadrics, we use full rank identity matrices, aiming at preserving the original shape of the inserted pattern elements. A small detail helps at this point to improve the quality of the results: At the boundary between prestreched and unstreched geometry, the elastic deformation model tends to produce artifacts. Therefore, we set different error quadrics for pairs of nodes that connect across normal geometry and sliding docker areas. We use a quadric $\mathbf{I} - \mathbf{t}\mathbf{t}^\top$ in order to make the boundary slidable, not diffusing the errors introduced by the stress discontinuity.

Assembling a new shape by inserting stretched patterns creates shapes that are only C^0-continuous at the boundaries. The elastic deformation model will aim at undoing the deformation, but the subspace model cannot represent high frequency details so that a visible artifact at the boundary remains. In order to avoid this problem, we need to make sure that the new base shape that we create is actually smooth and the deformation is low frequency. Therefore, we use a windowing function $g(x)$ in the direction of the motion. We transition from the constant stretch factor of 1 to a different stretch s using a smooth step function. We employ a cosine step function $(1 - 0.5s \cos x)$ to transition from stretch 1 to a new constant stretch of s, and a similar cosine step leads back to 1. This function can be integrated analytically (to obtain the positions, rather than their derivatives) and inverted so that we can compute the inner stretch s that makes all instances fit into the placeholder. We fix the support of the smooth steps to always cover a support of at least $2h$ each, thereby creating a low-frequency distortion that remains within the Nyquist limit of the deformation model.

6.5 Implementation and Results

We have implemented the described shape editing system in C++ and evaluated it on a standard PC with an Intel Core-2 Quad CPU with 2.6GHz cores and 8GB of RAM. Our implementation is single-threaded. As benchmark data, we have collected a number of models from well-known commercial 3D model libraries. We use models from the Digimation Archive (www.digimation.com) and from Dosch Design (www.doschdesign.com). We also include an example from [Kraevoy et al., 2008]. For models with large triangles, we perform one or more 1:4 subdivision steps to obtain a sufficiently densely sampled mesh such that even elastic deforma-



(a) original object (b) no elasticity (c) no local constraints

(d) no global constraints (e) full energy

Figure 6.8: *The effect of individual energy terms in the variational deformation framework. Leaving out any single energy term leads to artifacts. Local constraints are important to distribute stretch into regions that are less affected; in (c) stretch is distributed uniformly which leads to artifacts e.g. in the tank caps. Without global constraints bending artifacts appear near the ladder (d).*

tion with bending can be represented.

Figures 6.1, 6.9, 6.10, 6.12 and 6.13 show a number of example models that have been edited using our approach. Please refer to the accompanying video for a demonstration of interactive editing. The deformation results produced by the technique are quite plausible; for many of the examples, it would be challenging to identify the original model without the highlighting. Some minor artifacts can be seen due to small-scale irregularities in the input geometry (see discussion in the next section), which cause some patterns to only be detected in chunks that are not constrained simultaneously, leading to a small amount of global deformation.

The blue examples use only the elastic energy term (Equation 6.2), with discrete relaxation still enabled. On the one hand, these examples demonstrate that the pattern-aware continuous deformation model is crucial for maintaining the structure of the original model. On the other hand, the severe deformations shown in these examples serve as a stress test for sliding dockers. Despite strong bending, our approach reliably adapts the repetition count of the patterns without any seams, discontinuities, gaps, or similar artifacts. In some models, the triangulation becomes visible; this could be resolved by a more elaborate adaptive mesh subdivision scheme.

Limitations of the implementation: Our implementation uses only translational patterns. Continuous rotational pattern are preserved implicitly by the local constraints, but global constraints would probably improve the results. Furthermore, discrete constraints are only imposed on single sliding docker groups and do not connect multiple groups that form the same pattern.

Parameters: Our algorithm is not very sensitive to parameter settings, and we

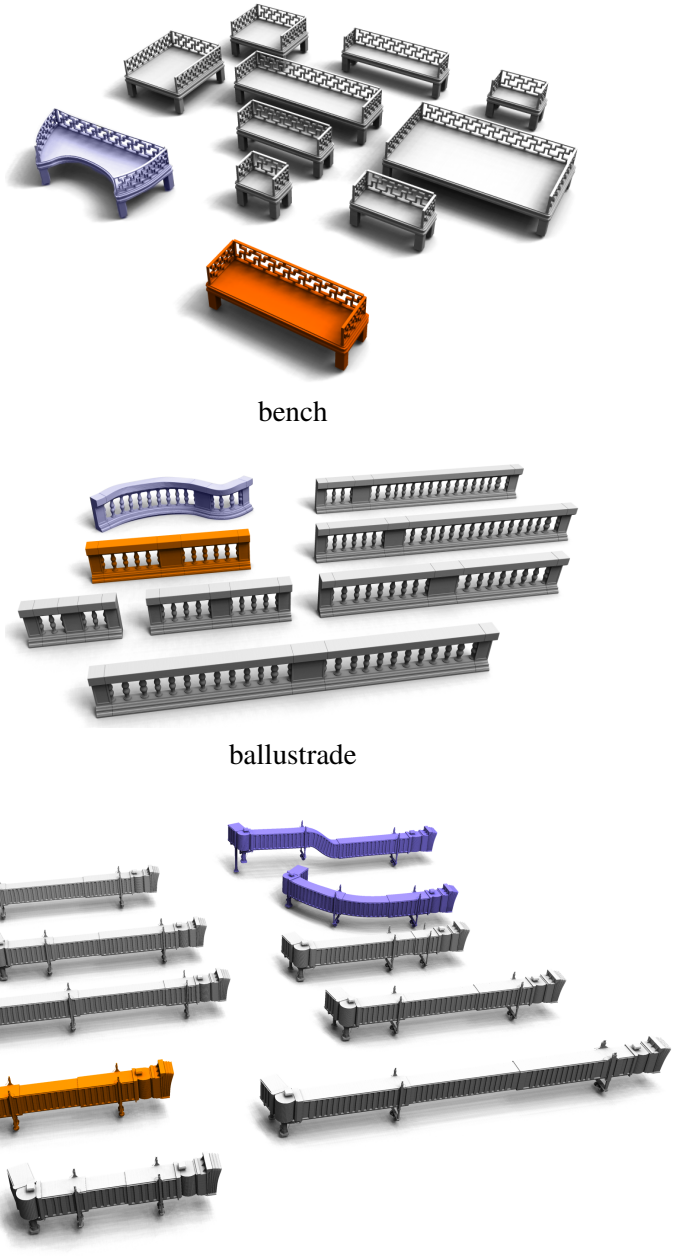
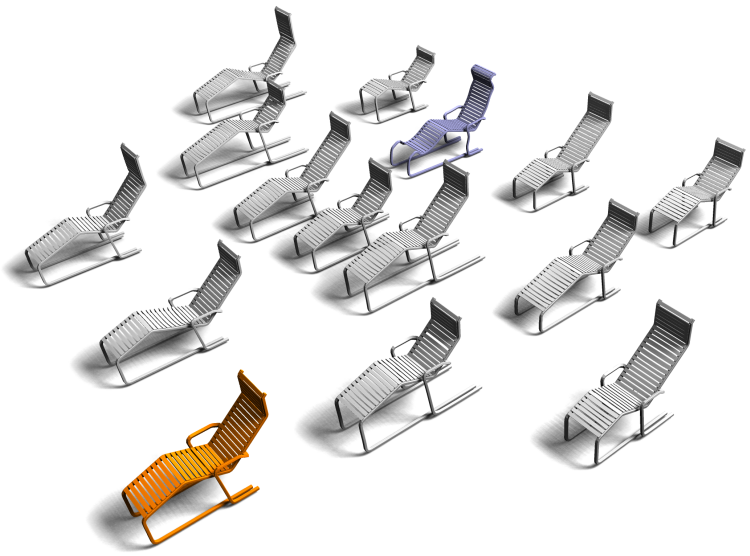
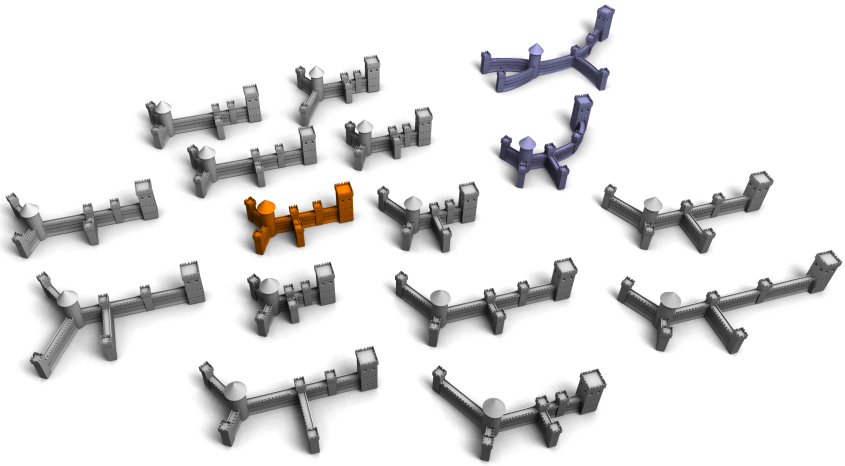


Figure 6.9: *Results of our pattern-aware shape editing method. The original input is shown in orange and editing results are shown in grey and blue. The repetition counts of discrete patterns were automatically adapted by the framework in both grey and blue results. For the blue models, pattern preservation constraints were disabled and only the elastic energy was used, in order to allow for more severe deformation.*

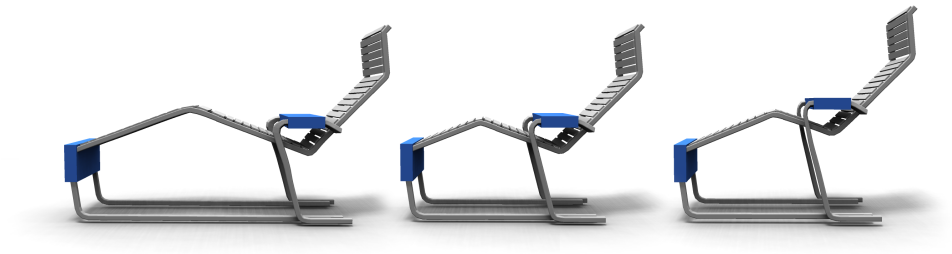


canvas chair



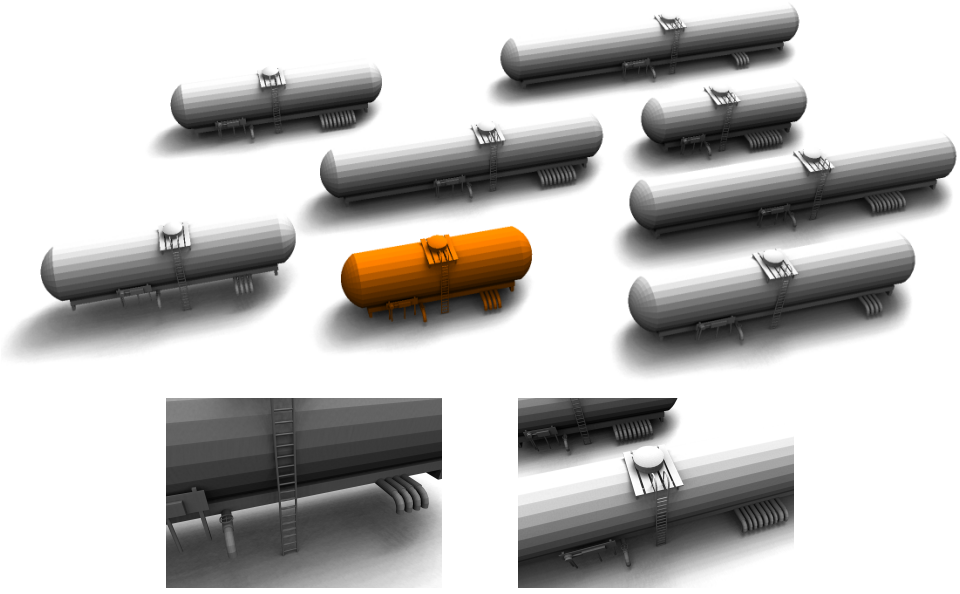
castle

Figure 6.10: *Results for more complex shapes. Both the chair and the castle contain multiple patterns with linear independent directions. See also Figure 6.11 for an illustration of stretch distribution.*



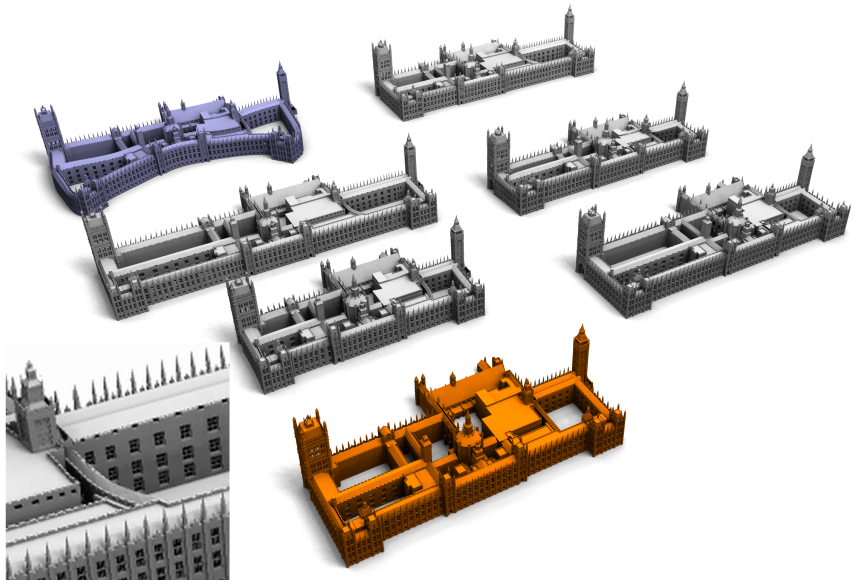
stretched horizontally original stretched vertically

Figure 6.11: *Stretch distribution in the canvas chair example: Horizontal stretch in the upper part is mostly distributed on two patterns between the user constraints. Here, a single pattern would not be able to express horizontal stretch, however, the combination of both yield plausible results. Similarly, another combination of the same patterns compensates vertical stretch in the shown configuration.*

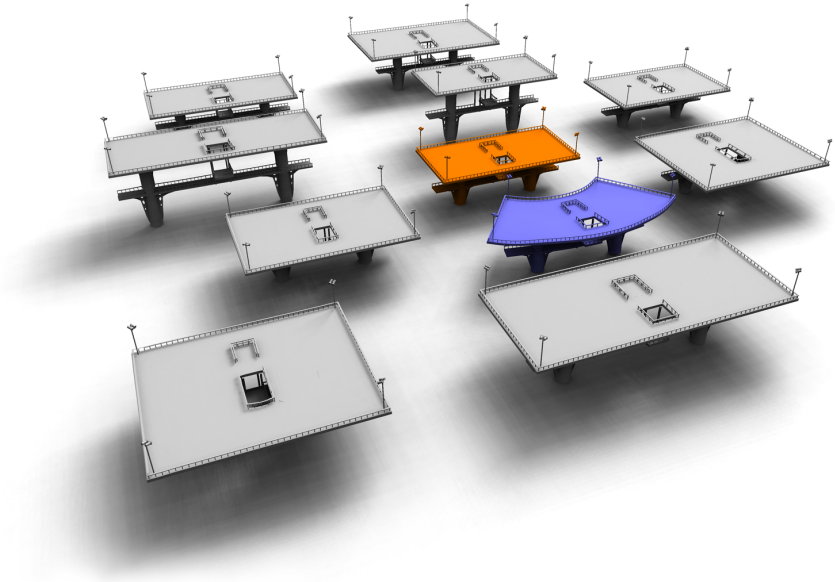


extended ladder bending artifact

Figure 6.12: *Different variations for oiltank data set. Please note how stretch is distributed mostly within the cylindrical parts of the tank. Both caps and the ladder are relatively unaffected. However, some minor artifacts can appear due to the least-squares optimization scheme and some undetected patterns (visible in the slight bending of the ladder).*



houses of parliament



platform

Figure 6.13: *Results for large objects. The object "houses of parliament" is the most complex example that we tested and contains a large number of regular patterns. Our method can successfully exploit these patterns to reduce stretch. Because of the limited resolution of the deformation field, nearby patterns can produce locking constraints and result in distortions shown in the detail view.*

mostly use default parameters everywhere. Only one parameter has a strong effect on the quality of the results: For inaccurate models, we need to set the error threshold for matching "symmetric" line features. For objects with gross irregularities in the intended patterns, care must be taken not to set excessively large thresholds such that unrelated elements are matched. Two additional parameters are adapted manually for efficiency reasons. For complex models (such as the "houses of parliament"), we lower the minimum size of relevant features (default: $2.5\%\ l(\mathcal{S})$) in order to detect finer-grained patterns. Furthermore, we increase the resolution of the subspace deformation model when necessary (the "houses of parliament" use $2.5\%\ l(\mathcal{S})$ instead of $5\%\ l(\mathcal{S})$, which leads to a reduced interactive frame-rate).

Timings: As shown in the video, editing can be done interactively for all presented models. The structure analysis in preprocessing takes up to a few minutes for each model, and prefactorization of the linear systems adds another ten seconds.

Effect of individual energy terms: In Figure 6.8, we show that all ingredients of the variational framework are necessary to obtain good results. Deactivating the elastic energy means that no stretch is diffused and no reasonable editing is possible. The local constraints are helpful as they provide a better base regularizer for object parts where no patterns are found. The global constraints are necessary to keep objects straight; without them, global bending cannot be prevented.

Comparison to related work: Figure 6.12 shows the "oil tank" model used by Kraevoy et al. [Kraevoy et al., 2008]. Our technique achieves comparable results, while detecting the stretch axes fully automatically (we do not need to align the model with the global coordinate axes). Since we only penalize structural deviations in a least-squares sense, we obtain a minimal amount of residual bending (see the close-up; in particular, the ladder yields two separate sliding docker groups). The "castle" example in Figure 6.10 demonstrates that our approach is more general: The castle can be stretched in different, non-orthogonal directions, which are determined fully automatically by our pattern-aware structure model. This is not possible with the previous approach (see also the video and the "canvas chair" example). In comparison to the original docking site method presented in the previous chapter, our new approach can adapt the discrete structure of the model in real-time in response to continuous free-form deformation, instead of being driven by manually specified rigid shape operations. Furthermore, sliding dockers are found in examples where the analysis technique of our previous method fails to detect global cuts, such as the arches in Figure 6.1.

6.6 Discussion

We have presented a structure-aware deformation technique that uses the elementary assumption of preserving regular patterns, which we model as 1-parameter partial symmetry groups. We have developed a variational optimization technique that preserves such structures in a least squares sense, while distributing the remaining stretch uniformly. In addition, we introduced *sliding dockers* that allow

the technique to fully automatically insert or delete repeated elements in discrete patterns in order to minimize distortions due to free-form deformation. Furthermore, we have presented a numerical framework that uses a subspace formulation with prefactored linear systems to implement the presented approach efficiently and robustly in a real-time system.

One limitation of the current approach is the handling of small-scale irregularities in the input 3D model. Objects that appear perfectly regular often have geometric inconsistencies because the artist did not accurately align parts of the original model; errors of 10% are not uncommon. While we can compensate for this by introducing a small numerical threshold in the pattern detection algorithm, irregularities in the input can cause the shape analysis stage to overlook visually salient patterns. For the "houses of parliament" model, we had to manually adjust the original geometry in one place to repair a single discrete pattern. Making the analysis stage more robust to approximate regularity is a natural avenue for future work, possibly using a feature graph matching approach as in Chapter 4. A further limitation, and another interesting avenue for future work, is that all sliding docker groups currently need to be mutually disjoint. Handling overlapping and hierarchical patterns could extend the applicability of our method to more complex structures.

Another limitation of the presented approach is that it only handles rigid symmetries. Many interesting structural regularities are not easily captured by rigid patterns. In future work, we would like to investigate structure models that utilize more invariant notions of similarity, possibly incorporating scaling and intrinsic isometries. Finally, the presented work focuses on 1-parameter patterns. Future work could pursue a more comprehensive representation of the algebraic structure of partial symmetries for free-form shape deformation. This direction can meaningfully advance the capabilities of interactive shape editing tools.

7 Conclusions

Dunkel war's, der Mond schien helle,
Als ein Wagen mit Blitzesschnelle
Langsam um die Ecke fuhr...

unkown author

In this chapter, we give a summary of the main contributions of the thesis and discuss limitations, insights, and potential directions for future work. The thesis is concluded by a few personal remarks of the author.

Part I

In the first part of the thesis, we presented two novel feature detection methods for unstructured point cloud data in the context of correspondence analysis and symmetry detection. The first feature detector considers points as features that have a local neighborhood with minimal slippage. As a result, we obtain feature points that are particularly well-suited for local alignment since we optimize for maximally constrained neighborhoods with respect to the auto-alignment problem i.e. registration of a local neighborhood with itself. We showed that our method produces stable keypoints comparable to and in some scenarios even better than state-of-the-art methods. In contrast to previous methods, we usually obtain a larger set of stable keypoints. Further, we demonstrated the applicability of our method to global registration for rigid and deformable objects as well as symmetry detection. For symmetry detection, however, the results leave room for improvement: The combination of heavy loss of information in the feature representation and a rather strict strategy for finding building blocks lead to recognition problems in noisy scanner data.

We improved upon the previous attempt to detect symmetries significantly by two fundamental changes: First, we considered lines as features instead of keypoints and second, an improved decomposition strategy. The proposed technique

to extract line features is significantly faster than the presented keypoint extraction method while the resulting line features carry highly descriptive information of the underlying surface. Keypoints can be seen as implicitly encoded in the line feature representation by two neighboring line feature with different directions. These implicit keypoints can be used to generate candidate transformations between two pairs of feature lines and thereby bootstrap the randomized search strategy in a promising direction.

Further, we realized that partial symmetry detection is substantially different from building block decomposition. For a robust decomposition technique, we have to consider partial symmetry *before* we decompose the shape into parts; the keypoint-based symmetry detection technique [Berner et al., 2008] tries to combine both into one step, symmetry detection and decomposition, which results in rejection of many symmetric instances. Due to noise, we will always lose some features and hallucinate other features, that are not present in the data. Consequently, we cannot demand a consistent set of features across all instances of a building block and the algorithm has to adapt to that. In the improved algorithm, we first detect a set of partial symmetries without considering their interrelationship; we only gather pairwise evidence in the line feature representation individually for each potential instance. Here, the line feature representation provides a fast verification test (the set of line features is usually two orders of magnitude smaller than the original point data while still being highly discriminative). In the following step, we extract the most common building blocks directly from the original point data. This way we avoid to miss instances where some features are absent.

The presented approach is a major contribution of this thesis. It allows symmetry detection and building block decomposition on large datasets that contain many different symmetries without restrictions to special cases. Techniques that are based on transformation voting have serious difficulties here because the transformation space clutters and it becomes hard and at some point impossible to identify meaningful symmetries. In comparison to related work, our method can handle much larger data sets (up to two orders of magnitude more complex) as they occur in city scanning and is able to detect a large number of building blocks in quantities that have not been demonstrated before by existing methods. Our method works efficiently due to the line feature representation that allows rapid, reliable geometric matching.

A limitation of our approach is that we depend on feature lines. On some models we might not find a suitable set of features. This is a limitation that transformation voting approaches do not have. However, in the data we observed so far this was not an issue.

Based on the line feature representation, a large number of meaningful symmetry candidates are generated that are rejected in a later stage of the algorithm due to our decomposition strategy. Again, we are limited by performing symmetry detection and shape decomposition together in one step. Decoupling the detection of partial symmetry from the decomposition process might result in a much more robust and effective way to retrieve building blocks. This is related to the method

presented in Chapter 5 where we compute partial symmetries and find shape operations defined by global cuts through symmetric area. Cutting a shape with all docking sites simultaneously could lead to a set of canonical building blocks, however, only for perfect symmetries. The main challenges of building block decomposition for general scanner data are caused by noise, outliers, and holes in the data. Therefore, finding canonical building blocks in scanner data is still an interesting problem for future work.

Further, it would be rewarding to change the notion of strictly disjoint building blocks to a relaxed definition that allows overlapping parts, nested structures, and regular patterns because this would provide us with more information and yields inherent properties that we can exploit in the detection process.

The presented method can be characterized as unsupervised symmetry detection where only a small set of data-dependent parameters are provided by the user. As an alternative, semi-supervised techniques take a small number of training examples as additional input and try to learn how to recognize instances of this class in the data. It is not clear yet how one can formulate a learning-based symmetry detection algorithm, however, first results in this direction show potential [Sunkel et al., 2011]. In future work, we would like to combine canonical low-level features, such as the presented line features, with machine learning techniques in order to detect more general types of symmetries.

Part II

In the second part of the thesis, we presented two methods for example-based 3D content creation by utilization of their symmetry structure. First, we discovered that partial symmetries yield docking sites that allow us to replace geometry and thus produce variations of the exemplar while providing strong guarantees of the resulting shapes; all output shapes are r-similar to the input shape. The concept of docking sites is derived directly from partial symmetry, and represents the first theoretically sound framework for synthesizing r-similar shapes. This concept is a major contribution of this thesis and defines a first step in the relatively unexplored area of inverse procedural modeling of 3D shapes. We developed a method that systematically extracts docking sites from an exemplar and computes a shape grammar that produces r-similar shapes. Our method can handle large triangles meshes as well as point cloud data and offers semi-automatic modeling and fully automatic creation of random variations.

A major advantage of our method is the guarantee that every output object will be r-similar to the input; a replacement operation will always lead to a complete model since we enforce global partitioning in the docking site extraction. This strategy, however, leads to a fundamental restriction of the space of models that we can generate with our technique. In future work, we would like to overcome this limitation and extend our framework to cover the whole space of r-similar shapes. We believe that by cutting the surface at all docking sites simultaneously results in

a canonical set of dockers that span the complete space of r-similar objects.

Currently, our technique is limited to a single exemplar as input. However, for inverse procedural modeling it would be useful to take multiple input examples into account in order to learn meaningful subspaces out of the overall space of constructible models. Another exciting direction for future work would be to generalize the concept of docking sites to different notions of symmetry. The current method is not applicable to organic structures. Consequently, we would like to define a special kind of symmetry that captures organic structures. In combination with multiple input examples, one could build a powerful tool for inverse procedural modeling of organic content like plants that only requires a set of input examples.

In the second chapter of Part II, we proposed a new type of deformation regularization that maintains the inherent regularities of a shape and thus preserves its structure. We developed a variational model that is derived by considering a shape as a collection of 1-parameter symmetry groups. As mentioned above, the method presented in Chapter 5 is restricted to global cuts through the object. Here, we extended the concept of docking sites to sliding dockers that allow local changes in the repetition count of a pattern without the requirement to find a global cut. Further, we formulated all constraints of the deformation model, including discrete changes of regular patterns, in such a way that the system matrix is constant which allows fast solving using prefactorization.

There are some limitations of the described method that could be addressed in future work. In many 3D shapes we find nearly regular patterns; artists often create patterns manually, e.g., via moving copy & paste blocks of geometry. Visually, these patterns appear perfectly regular but a closer look reveals their imperfection. By relaxation of exact regular patterns to nearly regular patterns the method would be applicable to much larger range of objects. The same is true for approximate symmetries: Sometimes the geometry of pattern elements varies a bit but we would still think of them as a regular pattern. However, allowing more variability also introduces the chance of possible artifacts due to ambiguities. Therefore, this relaxation might be more than just changing the thresholds.

Similar to the original docking site technique, sliding dockers are currently restricted to rigid symmetries; here, translational 1-parameter patterns. A straightforward extension would be to include rotational or helical patterns in the model. And of course, it would also be interesting to incorporate 2-parameter and 3-parameter patterns in the deformation model. However, a much more interesting direction of future work, also in the context of this whole thesis, is to define symmetry in organic objects and to extend the method to organic patterns since they also appear frequently in natural objects. Most organic objects (e.g. plants, animals, landscapes) are created by a complex interplay of different physical, chemical, and biological processes. Intuitively, these objects follow rules that one could describe in a formal model. However, finding a suitable model that describes a class of objects is a hard task, even for a specific subclass of objects. A major challenge for future research will be to develop models that capture the structure of general or specific

objects. Additionally, a model has to be compact enough to allow computational efficiency, e.g., in the detection process. This problem is not specific for organic structures but also appears in man-made shapes such as buildings. The methods presented in Part II use rather limited models in comparison to the full range of possibilities in manufacturing or architecture. Research of more expressive ways to capture structure will be one of the fundamental problems in the future direction of content creation and shape understanding.

Personal Remarks

In the beginning of this thesis, we had a clear picture in mind of how symmetry can be detected and used to modify shapes for high-quality content generation. While the exact shape of our plan was still hiding in the mist of future work ahead of us, we were quite certain about the elementary structure of the problem. Intuition and basic knowledge about symmetry had forged a vague idea that we just had to investigate in more detail and decent engineering would solve all arising problems. However, as time went by, we discovered that our view of the problem was fundamentally wrong and the certainty of our plan vanished. The reason was simple: We focused too much on symmetry and somehow ignored the opposite. We were trapped by the idea of building blocks.

The breakthrough came at the point where we considered all aspects of partial symmetry - symmetric *and* non-symmetric parts. This change of perspective presented symmetry in a different light and finally led to the concept of docking sites. Our intuition from the beginning was not entirely wrong but lacked deeper understanding. This experience told me that no matter how confident and plausible a plan appears to be, it is always beneficial to take a step back and look from another point of view. Also sometimes, you need to think about the contrary.

Bibliography

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

Aliaga, D., Rosen, P., and Bekins, D. (2007). Style grammars for interactive visualization of architecture. *IEEE Trans. Vis. Comp. Graph.*, 13(4):786–797.

Allen, B., Curless, B., and Popović, Z. (2003). The space of human body shapes: reconstruction and parameterization from range scans. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 587–594, New York, NY, USA. ACM.

Alt, H., Mehlhorn, K., Wagener, H., and Welzl, E. (1988). Congruence, similarity and symmetries of geometric objects. *Discrete Comput. Geom.*, 3(3):237–256.

Anguelov, D., Srinivasan, P., Pang, H.-C., Koller, D., Thrun, S., and Davis, J. (2004). The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces. In *NIPS*.

Atallah, M. J. (1985). On symmetry detection. *IEEE Trans. Computers*, pages 663–666.

Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.

Barla, P., Breslav, S., Thollot, J., Sillion, F., and Markosian, L. (2006). Stroke pattern analysis and synthesis. *Computer Graphics Forum*, 25(3).

Ben-Chen, M., Butscher, A., Solomon, J., and Guibas, L. J. (2010). On discrete killing vector fields and patterns on surfaces. *Comput. Graph. Forum*, 29(5):1701–1711.

Ben-Chen, M., Weber, O., and Gotsman, C. (2009). Variational harmonic maps for space deformation. *ACM Transactions on Graphics*, 28(3).

Bendels, G. H., Klein, R., and Schilling, A. (2003). Image and 3d-object editing with precisely specified editing regions. In *Vision, Modeling and Visualisation 2003*, pages 451–460.

Berner, A., Bokeloh, M., Wand, M., Schilling, A., and Seidel, H.-P. (2008). A graph-based approach to symmetry detection. In *Proc. Symp. Point-Based Graphics 2008*.

Berner, A., Wand, M., Mitra, N. J., Mewes, D., and Seidel, H.-P. (2011). Shape analysis with subspace symmetries. *Computer Graphics Forum*, 30(2). Proceedings Eurographics 2011.

Besl, P. J. and Mckay, N. (1992). A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2).

Bhat, P., Ingram, S., and Turk, G. (2004). Geometric texture synthesis by example. In *Symp. Geometry Processing*.

Blanz, V. and Vetter, T. (1999). A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 187–194.

Bokeloh, M., Berner, A., Wand, M., Schilling, A., and Seidel, H.-P. (2008). Slippage features. Technical Report WSI-2008-03, University of Tübingen, WSI/-GRIS.

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

Bokeloh, M., Wand, M., Koltun, V., and Seidel, H.-P. (2011). Pattern-aware shape deformation using sliding dockers. *ACM Transactions on Graphics*, 30(6).

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

Botsch, M. and Kobbelt, L. (2004). An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics*, 23(3):630–634.

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

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

Cabral, M., Lefbvre, S., Dachsbacher, C., and Drettakis, G. (2009). Structure-preserving reshape for textured architectural scenes. *Computer Graphics Forum*, 28(2).

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

Chaudhuri, S., Kalogerakis, E., Guibas, L., and Koltun, V. (2011). Probabilistic reasoning for assembly-based 3d modeling. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pages 35:1–35:10.

Chaudhuri, S. and Koltun, V. (2010). Data-driven suggestions for creativity support in 3d modeling. In *ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA '10, pages 183:1–183:10.

Chen, L. and Meng, X. (2009). Anisotropic resizing of model with geometric textures. In *Conf. on Geometric and Physical Modeling (SPM)*, pages 289–294, New York, NY, USA. ACM.

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

Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619.

Coquillart, S. (1990). Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *Proc. Siggraph*, pages 187–196.

Efros, A. A. and Leung, T. K. (1999). Texture synthesis by non-parametric sampling. In *Proc. Int. Conf. Comp. Vision*.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395.

Fisher, M., Savva, M., and Hanrahan, P. (2011). Characterizing structural relationships in scenes using graph kernels. *ACM Trans. Graph.*, 30:34:1–34:12.

Frome, A., Huber, D., Kolluri, R., Bulow, T., and Malik, J. (2004). Recognizing objects in range data using regional point descriptors. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Funkhouser, T., Kazhdan, M., Shilane, P., Min, P., Kiefer, W., Tal, A., Rusinkiewicz, S., and Dobkin, D. (2004). Modeling by example. *ACM Trans. Graph.*, 23(3).

Gal, R. and Cohen-Or, D. (2006). Salient geometric features for partial shape matching and similarity. *ACM Trans. Graph.*, 25(1):130–150.

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

Gal, R., Sorkine, O., Mitra, N., and Cohen-Or, D. (2009). iwires: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph.*, 28(3).

Gatzke, T., Grimm, C., Garland, M., and Zelinka, S. (2005). Curvature maps for local shape comparison. In *In Shape Modeling International*, pages 244–256.

Gelfand, N. and Guibas, L. J. (2004). Shape segmentation using local slippage analysis. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 214–223, New York, NY, USA. ACM.

Gelfand, N., Ikemoto, L., Rusinkiewicz, S., and Levoy, M. (2003). Geometrically stable sampling for the icp algorithm. In *Proc. Int. Conf. 3D Digital Imaging and Modeling*.

Gelfand, N., Mitra, N. J., Guibas, L. J., and Pottmann, H. (2005). Robust global registration. In *Proc. Symp. Geometry Processing*, pages 197–206.

Gortler, S. J., Grzeszczuk, R., Szeliski, R., and Cohen, M. F. (1996). The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA. ACM Press.

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

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

Hart, J. and Flynn, O. C. P. (1997). Similarity hashing: A computer vision solution to the inverse problem of linear fractals. *Fractals*, 5:35–50.

Hasler, N., Stoll, C., Sunkel, M., Rosenhahn, B., and Seidel, H.-P. (2009). A Statistical Model of Human Pose and Body Shape. *Comput. Graph. Forum*, 28(2):337–346.

Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., and Salesin, D. H. (2001). Image analogies. In *Proc. Siggraph 2001*, pages 327–340.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1992). Surface reconstruction from unorganized points. In *Proc. Siggraph 1992*.

Huang, J., Shi, X., Liu, X., Zhou, K., Wei, L.-Y., Teng, S.-H., Bao, H., Guo, B., and Shum, H.-Y. (2006a). Subspace gradient domain mesh deformation. *ACM Trans. Graph.*, 25(3):1126–1134.

Huang, Q., Mech, R., and Carr, N. (2009). Optimizing structure preserving embedded deformation for resizing images and vector art. In *Pacific Graphics*.

Huang, Q.-X., Flöry, S., Gelfand, N., Hofer, M., and Pottmann, H. (2006b). Reassembling fractured objects by geometric matching. *ACM Trans. Graphics*, 25(3):569–578.

Huber, D. F. and Hebert, M. (2001). Fully automatic registration of multiple 3D data sets. In *IEEE Computer Society Workshop on Computer Vision Beyond the Visible Spectrum (CVBVS 2001)*.

Ijiri, T., Měch, R., Igarashi, T., and Miller, G. (2008). An example-based procedural system for element arrangement. *Computer Graphics Forum*, 27(3).

Johnson, A. E. and Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3d scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(5):433–449.

Joshi, P., Meyer, M., DeRose, T., Green, B., and Sanocki, T. (2007). Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26.

Ju, T., Schaefer, S., and Warren, J. (2005). Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24:561–566.

Kalogerakis, E., Hertzmann, A., and Singh, K. (2010). Learning 3D Mesh Segmentation and Labeling. *ACM Transactions on Graphics*, 29(3).

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

Kazhdan, M., Funkhouser, T., and Rusinkiewicz, S. (2003b). Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Proc. Symp. Geometry Processing*.

Kim, V., Lipman, Y., Chen, X., and Funkhouser, T. (2010). Mobius transformations for global intrinsic symmetry analysis. *Computer Graphics Forum (Symposium on Geometry Processing)*, 29(5).

Kraevoy, V., Julius, D., and Sheffer, A. (2007). Shuffler: Modeling with interchangeable parts. In *Pacific Graphics 2007*.

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

Kwatra, V., Schödl, A., Essa, I., Turk, G., and Bobick, A. (2003). Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286.

Lagae, A., Dumont, O., and Dutré, P. (2005). Geometry synthesis by example. In *Conf. Shape Modeling and Applications*.

Lai, Y.-K., Hu, S.-M., Gu, D. X., and Martin, R. R. (2005). Geometric texture synthesis and transfer via geometry images. In *Symp. Solid and Physical Modeling*, pages 15–26.

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

Leordeanu, M. and Hebert, M. (2005). A spectral technique for correspondence problems using pairwise constraints. In *International Conference of Computer Vision (ICCV)*, volume 2, pages 1482 – 1489.

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

Li, X., Guskov, I., and Barhak, J. (2006). Robust alignment of multi-view range data to cad model. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, page 17, Washington, DC, USA. IEEE Computer Society.

Lipman, Y., Chen, X., Daubechies, I., and Funkhouser, T. (2010). Symmetry factored embedding and distance. *ACM Trans. Graph.*, 29:103:1–103:12.

Lipman, Y., Levin, D., and Cohen-Or, D. (2008). Green coordinates. *ACM Trans. Graph.*, 27.

Liu, L., Zhang, L., Xu, Y., Gotsman, C., and Gortler, S. (2008). A local/global approach to mesh parameterization. *Computer Graphics Forum*, 27(5):1495–1504.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. In *Int. J. Computer Vision*, volume 20, pages 91–110.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99.

Loy, G. and Eklundh, J. (2006). Detecting symmetry and symmetric constellations of features. In *Proc. Europ. Conf. Computer Vision*, pages 508–521.

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

Merrell, P. (2007). Example-based model synthesis. In *Symp. Interactive 3D Graphics and Games*, pages 105–112.

Merrell, P. and Manocha, D. (2008). Continuous model synthesis. *ACM Trans. Graph.*, 27(5):1–7.

Merrell, P., Schkufza, E., and Koltun, V. (2010). Computer-generated residential building layouts. In *ACM SIGGRAPH Asia 2010 papers*, SIGGRAPH ASIA '10, pages 181:1–181:12.

Merrell, P., Schkufza, E., Li, Z., Agrawala, M., and Koltun, V. (2011). Interactive furniture layout using interior design guidelines. *ACM Trans. Graph.*, 30:87:1–87:10.

Mezger, J., Thomaszewski, B., Pabst, S., and Straßer, W. (2008). Interactive physically-based shape editing. In *Symposium on Solid and Physical Modeling*, pages 79–89. ACM.

Miller, W. (1972). *Symmetry Groups and their Applications*. Academic Press.

Mitra, N. J., Bronstein, A., and Bronstein, M. (2010). Intrinsic regularity detection in 3d geometry. In *Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III*, ECCV'10, pages 398–410, Berlin, Heidelberg. Springer-Verlag.

Mitra, N. J., Gelfand, N., Pottmann, H., and Guibas, L. (2004). Registration of point cloud data from a geometric optimization perspective. In *Symp. Geometry Processing*.

Mitra, N. J., Guibas, L., Giesen, J., and Pauly, M. (2006a). Probabilistic fingerprints for shapes. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 121–130, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Mitra, N. J., Guibas, L., and Pauly, M. (2007). Symmetrization. In *ACM Transactions on Graphics*, volume 26.

Mitra, N. J., Guibas, L. J., and Pauly, M. (2006b). Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph.*, 25(3):560–568.

Mitra, N. J. and Pauly, M. (2008). Symmetry for architectural design. In *Advances in Architectural Geometry*, pages 13–16.

Moravec, H. P. (1981). Rover visual obstacle avoidance. In *The 7th International Joint Conference on Artificial Intelligence, Vancouver, British Columbia*, pages 785–790. IJCAI.

Müller, M., Dorsey, J., McMillan, L., Jagnow, R., and B., C. (2002). Stable real-time deformations. In *Proc. Symp. Computer Animation (SCA)*, pages 49–54.

Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Gool, L. V. (2006). Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623.

Müller, P., Zeng, G., Wonka, P., and Gool, L. V. (2007). Image-based procedural modeling of facades. *ACM Trans. Graph.*, 26(3).

Neubert, B., Franken, T., and Deussen, O. (2007). Approximate image-based tree-modeling using particle flows. *ACM Trans. Graph.*, 26(3).

Nguyen, M. X., Yuan, X., and Chen, B. (2005). Geometry completion and detail generation by texture synthesis. *The Visual Computer*, 21(9–10):669–678.

Novotni, M. and Klein, R. (2003). 3d zernike descriptors for content based shape retrieval. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 216–225, New York, NY, USA. ACM.

Ohtake, Y., Belyaev, A., and Seidel, H.-P. (2004). Ridge-valley lines on meshes via implicit surface fitting. In *Proc. Siggraph*, pages 609–612.

Ovsjanikov, M., Sun, J., and Guibas, L. (2008). Global intrinsic symmetries of shapes. In *Eurographics Symposium on Geometry Processing (SGP)*.

Parish, Y. I. H. and Müller, P. (2001). Procedural modeling of cities. In *Proc. Siggraph 2001*, pages 301–308.

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

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

Pauly, M., Mitra, N. J., Wallner, J., Pottmann, H., and Guibas, L. (2008). Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics*, 27(3).

Podolak, J., Shilane, P., Golovinskiy, A., Rusinkiewicz, S., and Funkhouser, T. (2006). A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3).

Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants*. Springer Verlag.

Raviv, D., Bronstein, A. M., Bronstein, M. M., and Kimmel, R. (2007). Symmetries of non-rigid shapes. In *Proc. Non-rigid Registration and Tracking (NRTL) workshop. See Proc. of International Conference on Computer Vision (ICCV)*.

Raviv, D., Bronstein, A. M., Bronstein, M. M., and Kimmel, R. (2009). Full and partial symmetries of non-rigid shapes. *International Journal of Computer Vision (IJCV)*.

Raviv, D., Bronstein, A. M., Bronstein, M. M., Kimmel, R., and Sapiro, G. (2010). Diffusion symmetries of non-rigid shapes. In *Proc. 3D Data Processing, Visualization and Transmission (3DPVT)*.

Rodrigues, O. (1816). De l'attraction des sphéroïdes. *Correspondence sur l'École Impériale Polytechnique 3 (3)*.

Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the ICP algorithm. In *Proc. 3rd Intl. Conf. 3D Digital Imaging and Modeling*, pages 145–152.

Rustamov, R. M. (2008). Augmented planar reflective symmetry transform. *Vis. Comput.*, 24(6):423–433.

Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226.

Schnabel, R., Wessel, R., Wahl, R., and Klein, R. (2008). Shape recognition in 3d point-clouds. In *Proc. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*.

Sederberg, T. W. and Parry, S. R. (1986). Free-form deformation of solid geometric models. In *Proc. Siggraph*, pages 151–160.

Sharf, A., Alexa, M., and Cohen-Or, D. (2004). Context-based surface completion. *ACM Trans. Graph.*, 23(3):878–887.

Simari, P., Kalogerakis, E., and Singh, K. (2006). Folding meshes: hierarchical mesh segmentation based on planar symmetry. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 111–119.

Snavely, N., Seitz, S. M., and Szeliski, R. (2006). Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH Conference Proceedings*, pages 835–846, New York, NY, USA. ACM Press.

Sorkine, O. and Alexa, M. (2007). As-rigid-as-possible surface modeling. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 109–116.

Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., and Seidel, H.-P. (2004). Laplacian surface editing. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184, New York, NY, USA. ACM.

Sumner, R. W., Schmid, J., and Pauly, M. (2007). Embedded deformation for shape manipulation. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 80, New York, NY, USA. ACM.

Sunkel, M., Jansen, S., Wand, M., Eisemann, E., and Seidel, H.-P. (2011). Learning line features in 3d geometry. In *Computer Graphics Forum (Proc. EUROGRAPHICS)*, volume 30.

Talton, J., Lou, Y., Lesser, S., Duke, J., Měch, R., and Koltun, V. (2011). Metropolis procedural modeling. *ACM Trans. Graphics*, 30(2).

Tan, P., Zeng, G., Wang, J., Kang, S. B., and Quan, L. (2007). Image-based tree modeling. *ACM Trans. Graph.*, 26(3).

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. (1987). Elastically deformable models. In *Proc. SIGGRAPH '87*, pages 205–214, New York, NY, USA. ACM.

Tevs, A., Bokeloh, M., Wand, M., Schilling, A., and Seidel, H.-P. (2009). Isometric registration of ambiguous and partial data. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009)*. IEEE Computer Society.

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

Toledo, S. (2003). Taucs: A library of sparse linear solvers. Tel-Aviv University, available online at http://www.tau.ac.il/ stoledo/taucs/.

Šťava, O., Beneš, B., Měch, R., Aliaga, D., and Krištof, P. (2010). Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum*, pages 665–674.

von Funck, W., Theisel, H., and Seidel, H.-P. (2006). Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125.

Wand, M., Jenke, P., Huang, Q.-X., Bokeloh, M., Guibas, L., and Schilling, A. (2007). Reconstruction of deforming geometry from time-varying point clouds. In *Proc. Symp. Geometry Processing*.

Wand, M. and Straßer, W. (2002). Multi-resolution rendering of complex animated scenes. *Computer Graphics Forum*, 21(3). Eurographics 2002.

Wang, Y., Xu, K., Li, J., Zhang, H., Shamir, A., Liu, L., Cheng, Z., and Xiong, Y. (2011). Symmetry hierarchy of man-made objects. *Computer Graphics Forum (Special Issue of Eurographics)*, 30(2):287–296.

Wolter, J. D., Woo, T. C., and Volz, R. A. (1985). Optimal algorithms for symmetry detection in two and three dimensions. *The Visual Computer*, 1(1):37–48.

Wonka, P., Wimmer, M., Sillion, F., and Ribarsky, W. (2003). Instant architecture. *ACM Trans. Graph.*, 22(3):669–677.

Wyngaerd, J. V. and Gool, L. V. (2002). Automatic crude patch registration: toward automatic 3d model building. *Comput. Vis. Image Underst.*, 87(1-3):8–26.

Xiao, J., Fang, T., Zhao, P., Lhuillier, M., and Quan, L. (2009). Image-based street-side city modeling. *ACM Trans. Graph.*, 28(5):1–12.

Xu, K., Zhang, H., Tagliasacchi, A., Liu, L., Li, G., Meng, M., and Xiong, Y. (2009a). Partial intrinsic reflectional symmetry of 3d shapes. *ACM Trans. Graph.*, 28:138:1–138:10.

Xu, W., Wang, J., Yin, K., Zhou, K., van de Panne, M., Chen, F., and Guo, B. (2009b). Joint-aware manipulation of deformable models. *ACM Trans. Graph.*, 28(3):1–9.

Yamany, S. M. and Farag, A. A. (2002). Surfacing signatures: An orientation independent free-form surface representation scheme for the purpose of objects registration and matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(8):1105–1120.

Yeh, Y.-T. and Měch, R. (2009). Detecting symmetries and curvilinear arrangements in vector art. *Computer Graphics Forum*, 28(2):707–716.

Zelinka, S. and Garland, M. (2006). Surfacing by numbers. In *Graphics Interface 2006*.

Zheng, Q., Sharf, A., Wan, G., Li, Y., Mitra, N. J., Cohen-Or, D., and Chen, B. (2010). Non-local scan consolidation for 3d urban scenes. *ACM Transactions on Graphics*, 29:94:1–94:9.

Zheng, Y., Fu, H., Cohen-Or, D., Au, O. K.-C., and Tai, C.-L. (2011). Component-wise controllers for structure-preserving shape manipulation. *Comput. Graph. Forum*, 30(2):563–572.

Zhou, K., Huang, J., Snyder, J., Liu, X., Bao, H., Guo, B., and Shum, H.-Y. (2005). Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph.*, 24(3):496–503.

Zhou, K., Huang, X., Wang, X., Tong, Y., Desbrun, M., and Baining Guo, H.-Y. S. (2006). Mesh quilting for geometric texture synthesis. *ACM Trans. Graph.*, 25(3):690–697.

Complete List of the Authors Publications

A. Tevs, A. Berner, M. Wand, I. Ihrke, M. Bokeloh, J. Kerber, H.-P. Seidel: Animation Cartography - Intrinsic Reconstruction of Shape and Motion. ACM Transaction on Graphics 2012 (conditionally accepted)

M. Bokeloh, M. Wand, V. Koltun, H.-P. Seidel: Pattern-Aware Shape Deformation Using Sliding Dockers. SIGGRAPH ASIA 2011

M. Bokeloh, M. Wand, H.-P. Seidel: A Connection between Partial Symmetry and Inverse Procedural Modeling. SIGGRAPH 2010

Jens Kerber, Martin Bokeloh, Michael Wand, Jens Krüger, Hans-Peter Seidel: Feature Preserving Sketching of Volume Data. International Workshop on Vision, Modeling and Visualization, 2010.

A. Berner, M. Bokeloh, M. Wand, A. Schilling, H.-P. Seidel: Generalized Intrinsic Symmetry Detection. MPI Informatik Tech Report MPI-I-2009-4-005, 2009.

A. Tevs, M. Bokeloh, M.Wand, A. Schilling, H.-P. Seidel: Isometric Registration of Ambiguous and Partial Data. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR '09), 2009.

M. Bokeloh, A. Berner, M.Wand, H.-P. Seidel, A. Schilling: Symmetry Detection Using Feature Lines. In: Computer Graphics Forum, Proc. Eurographics '09.

M. Wand, B. Adams, M. Ovsjanikov, A. Berner, M. Bokeloh, P. Jenke, L. Guibas, H.-P. Seidel, A. Schilling: Efficient Reconstruction of Non-rigid Shape and Motion from Real-Time 3D Scanner Data. In: ACM Transactions on Graphics 28(2), April 2009.

A. Berner, M. Bokeloh, M.Wand, A. Schilling, H.-P. Seidel: A Graph-Based Approach to Symmetry Detection. In: Proc. Symposium on Point-Based Graphics, 2008.

M. Bokeloh, A. Berner, M. Wand, A. Schilling, H.-P. Seidel: Slippage Features. Technical Report, WSI-2008-03, University of Tübingen, 2008.

M. Wand, A. Berner, M. Bokeloh, P. Jenke, A. Fleck, M. Hoffmann, B. Maier, D. Staneker, A.Schilling, H.-P. Seidel: Processing and Interactive Editing of Huge Point Clouds from 3D Scanners. In: Computers and Graphics, 32(2), 204-220, 2008.

M. Wand, A. Berner, M. Bokeloh, A. Fleck, M. Hoffmann, P. Jenke, B. Maier, D. Staneker, A.Schilling: Interactive Editing of Large Point Clouds. In: Proc. Symposium on Point-Based Graphics (PBG 07), 2007.

M. Wand, P. Jenke, Q. Huang, M. Bokeloh, L. Guibas, and A. Schilling: Reconstruction of Deforming Geometry from Time-Varying Point Clouds. In: Proc. 5th Eurographics Symposium on Geometry Processing, Barcelona, Spain, pp. 49-58, 2007.

P. Jenke, M. Wand, M. Bokeloh, A. Schilling, W. Strasser: Bayesian Point Cloud Reconstruction. In: Computer Graphics forum 25(3), 379-388 (Proc. Eurographics 2006), 2006.

M. Bokeloh, M. Wand: Hardware Accelerated Multi-Resolution Geometry Synthesis. In: Symposium on Interactive 3D Graphics and Games 2006.