Die approbierte Originalversion dieser Dissertation ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Edit Propagation using Geometric Analogies

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Paul Guerrero
Registration Number 0026761

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn Michael Wimmer

The dissertation has been reviewed by:

Michael Wimmer	Peter Wonka	Helmut Pottmann

Vienna, 19th September, 2014

Paul Guerrero

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Paul Guerrero
Sanatoriumstr. 21b 17/3, Vienna, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. September 2014

Paul Guerrero

Acknowledgements

The contributions presented in this thesis are not the result of isolated work, but rather the product of many discussions, meetings and cooperation with colleagues. Therefore, I would first like to thank my advisors and colleagues Stefan Jeschke, Michael Wimmer and Peter Wonka, who provided guidance and valuable input, and helped introduce me into the research field.

Thanks also go to my colleague Thomas Auzinger for his cooperation in one of the contributions and to all of my colleagues, for interesting discussions – not only about research – as well as the motivation of knowing people that have a goal similar to my own.

I would also like to thank Peter Wonka for giving me the opportunity to do research at the Visual Computing Center in KAUST for well over a year, which turned out to be a very nice work environment and the work done there greatly contributed to this thesis.

I am grateful to Michael Wimmer and Werner Purgathofer for providing and managing the research infrastructure at the Institute for Computer Graphics and Algorithms in Vienna.

Finally, I thank Jyh-Ming Lien, Evan Behar and Dirk Jan-Kroon for providing code that helped speed up the creation of some proof-of-concept implementations discussed in this thesis.

The work presented in this thesis was funded by the Vienna University of Technology, KAUST and the FWF Research Fund projects no. P 24352-N23 and P 24600-N23.

Abstract

Modeling complex geometrical shapes, like city scenes or terrains with dense vegetation, is a time-consuming task that cannot be automated trivially. The problem of creating and editing many similar, but not identical models requires specialized methods that understand what makes these objects similar in order to either create new variations of these models from scratch or to propagate edit operations from one object to all similar objects. In this thesis, we present new methods to significantly reduce the effort required to model complex scenes.

For 2D scenes containing deformable objects, such as fish or snakes, we present a method to find partial matches between deformed shapes that can be used to transfer localized properties such as texture between matching shapes. Shapes are considered similar if they are related by point-wise correspondences and if neighboring points have correspondences with similar transformation parameters. Unlike previous work, this approach allows us to successfully establish matches between strongly deformed objects, even in the presence of occlusions and sparse or unevenly distributed sets of matching features.

For scenes consisting of 2D shape arrangements, such as floor plans, we propose methods to find similar locations in the arrangements, even though the arrangements themselves are dissimilar. Edit operations, such as object placements, can be propagated between similar locations. Our approach is based on simple geometric relationships between the location and the shape arrangement, such as the distance of the location to a shape boundary or the direction to the closest shape corner. Two locations are similar of they have many similar relations to their surrounding shape arrangement. To the best of our knowledge, there is no method that explicitly attempts to find similar locations in dissimilar shape arrangements. We demonstrate populating large scenes such as floor plans with hundreds of objects like pieces of furniture, using relatively few edit operations.

Additionally, we show that providing several examples of an edit operation helps narrowing down the supposed modeling intention of the user and improves the quality of the edit propagation. A probabilistic model is learned from the examples and used to suggest similar edit operations. Also, extensions are shown that allow application of this method in 3D scenes. Compared to previous approaches that use entire scenes as examples, our method provides more user control and has no need for large databases of example scenes or domain-specific knowledge. We demonstrate generating 3D interior decoration and complex city scenes, including buildings with detailed facades, using only few edit operations.

Kurzfassung

Das Modellieren komplexer geometrischer Formen, wie Stadtszenen oder dichter Vegetation, ist eine zeitintensive Aufgabe, dessen Automatisierung nicht trivial ist. Das Problem viele ähnliche, aber nicht idente Modelle zu erstellen und bearbeiten, erfordert spezialisierte Methoden die verstehen welche Merkmale die Modelle ähnlich machen um entweder neue Variationen der Modelle zu erzeugen oder Bearbeitungsoperationen von einem Modell auf alle ähnlichen Modelle zu übertragen. In dieser Dissertation präsentieren wir Methoden um den Modellierungsaufwand komplexer Szenen signifikant zu reduzieren.

Für 2D Szenen die deformierbare Objekte, wie Fische oder Schlangen enthalten, präsentieren wir eine Methode um partielle Ähnlichkeiten der deformierten Objekte zu finden, die benutzt werden können um lokale Eigenschaften, wie zum Beispiel Texturen, zwischen ähnlichen Objekten zu übertragen. Objekte werden als ähnlich bezeichnet, wenn es zwischen ihnen punktweise Übereinstimmungen gibt und benachbarte Punkte Übereinstimmungen mit ähnlichen Transformationsparametern haben. Anders als bei vorherigen Methoden erlaubt uns diese Herangehensweise Übereinstimmungen zwischen stark deformierten Objekten zu finden, auch wenn diese partiell verdeckt oder durch wenig dichte oder ungleichmäßig verteilte geometrische Merkmale verbunden sind.

Für Szenen die aus Anordnungen mehrerer zweidimensionaler geometrischer Formen bestehen, wie zum Beispiel Hausgrundrisse, präsentieren wir Methoden um ähnliche Punkte innerhalb der Anordnungen zu finden, auch wenn die Anordnungen selbst keine Ähnlichkeiten aufweisen. Modellierungsoperationen, wie Objektplatzierungen, können zwischen ähnlichen Punkten propagiert werden. Unser Ansatz basiert auf einfachen geometrischen Relationen zwischen dem Punkt und der Anordnung von Formen, wie der Distanz zwischen Punkt und dem Rand einer Form, oder der Richtung zwischen Punkt und nächstgelegener Ecke einer Form. Zwei Punkte sind ähnlich, wenn sie viele ähnliche Relationen zur umgebenden Anordnung enthalten. Nach bestem Wissen existiert keine Methode die explizit versucht ähnliche Punkte in unähnlichen Anordnungen geometrischer Formen zu finden. Wir demonstrieren das Bestücken großer Szenen wie Gebäudegrundrisse mit hunderten von Objekten, wie zum Beispiel Möbelstücken, durch Anwendung relativ weniger Modellierungsoperationen.

Zusätzlich zeigen wir, dass die Angabe mehrerer Beispieloperationen hilft die Modellierungsabsicht des Benutzers zu konkretisieren und damit die Qualität der Propagierung von Operationen zu erhöhen. Die Beispieloperationen werden in einem probabilistisches Modell gelernt und dieses Modell wird benutzt um Vorschläge für ähnliche Operationen zu generieren. Zudem werden Erweiterungen der Methode vorgestellt, die eine Anwendung auf dreidimensionalen Szenen ermöglichen. Verglichen mit früheren Ansätzen, die ganze Szenen als Beispiele benutzen, erlaubt

unsere Methode direktere Kontrolle durch den Benutzer und ist nicht auf große Datenbanken von Beispielszenen oder Domänenspezifisches Wissen angewiesen. Wir demonstrieren das Generieren von dreidimensionaler Inneneinrichtung und komplexer Stadtszenen mit detaillierten Fassaden durch Anwendung weniger Modellierungsoperationen.

Contents

1 Introduction **1**
1.1 Geometric Analogies 2
1.2 Challenges 3
1.3 Dissertation Thesis 4
1.4 Contributions 4

2 State of the Art **7**
2.1 Shape Similarities 7
2.2 Shape Modeling 17

3 Geometric Analogies for Deformable 2D Shapes **27**
3.1 Introduction 27
3.2 Transformation Parameter Similarity 29
3.3 Algorithm Overview 32
3.4 Improving First-Order Correspondences 33
3.5 Region Matching 34
3.6 Results and Discussion 37
3.7 Summary 46

4 Single-Example Geometric Analogies for Shape Arrangements **47**
4.1 Introduction 47
4.2 Overview 49
4.3 Geometric Relationships 50
4.4 Pose Matching Importance 53
4.5 Matching Algorithm 54
4.6 Mirror Symmetries 57
4.7 Local Coordinate Systems 58
4.8 Application: Floor-Plan Editing 59
4.9 Results 61
4.10 Summary 74

5 Multi-Example Geometric Analogies for Shape Arrangements **75**
5.1 Introduction 75

5.2 Overview 76
5.3 Geometric Relationship Features 78
5.4 SL0 Regression 81
5.5 Finding Candidate Placements 82
5.6 Applications 84
5.7 Results 88
5.8 Summary 95

6 Summary and Conclusions 97
6.1 Key Contributions 97
6.2 Outlook 98
6.3 Conclusions 99

A Local Contour Descriptors to be Used With the TPS Method 101

B Transformation Space Visualizations 105

C Validity of the SL0 Kernel 111
C.1 Overview 111
C.2 Formal proof of validity of feature vector permutation 112
C.3 Alternative proof of SL0 kernel validity 113

Bibliography 117

Curriculum Vitae 125

CHAPTER 1

Introduction

The use of virtual two- and three-dimensional models is becoming more commonplace in the daily experience of most people. The cities and terrain in Google Earth, the vast worlds in Minecraft, the sprawling vegatation in the movie Avatar and the rise of augmented reality all contribute to the popularity and demand for large and detailed virtual models. Advances in rendering techniques and graphics hardware allow *displaying* complex models, but *creation* of these models is still a predominantly manual process: specialized 3D modeling programs give full control to expert users, but provide little automation. As the models continually increase in complexity, pure manual modeling becomes less feasible, and new solutions to automate repetitive but non-trivial tasks are required. When creating a large city, for example, manually modeling every house, every window and every door handle is not feasible. On the other hand, populating a city with cloned houses (i.e. exact replica) leads to extremely repetitive and unrealistic models. Methods are needed that obtain a high-level specification from the user and use this specification to facilitate the creation of complex and varied models.

Research efforts to facilitate modeling can be roughly categorized into three main directions. The first approach to obtain and represent the high-level specification of a model is called *procedural modeling* [89, 79, 95, 62, 93, 61, 80, 87]. In this approach the specification is encoded in a manually defined rule base or procedure. New models can be created from the specification by just providing a few parameters. However, the rule set or procedure that encodes a given specification may be non-trivial to find and might get extremely complicated and extensive for more complex models.

A second approach is called *analyze-and-edit* [36, 106, 12]. Here, the specification is encoded in a set of constraints that enforce the specification on all models; for example, constraints on structural relationships or on animations of model parts. A modeling application implementing this approach usually contains a manually defined library of domain-specific constraints. An analysis of a model determines where these constraints are applicable, and in subsequent modeling operations, the applied constraints are enforced by the application. Using this approach, the modeling process is more efficient because the constraints eliminate unwanted degrees of freedom. However, the constraints only depend on the model, not the user. They cannot learn and automate

operations done by the user, which makes them less usable for creating large and complex models such as city scenes.

Modeling by example [101, 28, 100] is a different approach that aims at learning the specification from a set of examples provided by the user. The learned specification may be encoded in different forms, for example as a rule base or as a statistical model, which can then be applied to modify or create new models. When using this approach, it is necessary to identify which parts of the examples are similar and which parts are different, as well as which parts of a new model are similar to the examples. For example, when learning the specification for a chair from multiple different example chairs, it is necessary to find corresponding parts in the examples, e.g., the back rests and the legs, and then determine which new back rests or legs are similar to these examples. Most current methods only work for relatively specific application domains, which are mainly defined by the choice of similarity. A large variety of similarity definitions exist [63, 7, 84, 92, 25, 81, 103, 16, 8, 73, 67, 66, 5, 58, 102, 18, 43], but even those do not cover all similarities that can be found by a human observer and that may be relevant for the modeling process. A main problem in modeling by example is finding a similarity definition that is general enough and works well for a given domain.

Edit-propagation methods [102, 43, 9] can be viewed as a specialization of modeling by example. Instead of giving examples for the desired result, examples for single edit operations are provided, and the learned specifications only describe single operations. Models are constructed by propagating several edit operations. Note that this approach does not require more work from the user than approaches that use complete models as example; the amount of edit operations required to construct the example models is the same. There are several advantages when using this approach. First, having access to every step of the modeling session provides information that can be used to improve the result. Second, there is immediate feedback after each edit operation, giving more direct control to the user. On the downside, the order of edit operations generally matters, requiring somewhat more expertise from the users. Edit propagation is often mentioned in literature as an application for dense mappings between shapes. While dense mappings can always be used to propagate edit operations, they are not a necessary requirement. Methods can be devised that do not explicitly establish a dense mapping. As in all modeling-by-example approaches, similarity plays a central role in edit-propagation methods. Propagated edits are applied to places that are similar to those of the original edits. In our work, we present new methods for edit propagation employing novel definitions of similarity that open up new and more general domains of application.

1.1 Geometric Analogies

When searching for similarities in a scene, people generally consider the *relations* between the scene objects as much as the properties of the objects themselves. A table and stack of plates alone do not make a dinner table, the arrangement of objects is important as well. The same principle applies to object parts. In a human face for example, the anatomical parts are expected to have certain geometric relations, that is, a certain *arrangement*. The arrangement of the parts is characteristic for the face as much as the properties of the parts themselves. Consider Figure 1.1a, where we show a portrait made out of vegetables by Italian painter Guiseppe Arcimboldo.

(a) (b) (c)

Figure 1.1: When searching for similarities in objects, the relations between individual parts is as important as the properties of the parts themselves. In the painting *Vertumnus* by Guiseppe Arcimboldo (a), only the arrangement of the vegetables suggests a face, the vegetables themselves only loosely resemble actual face parts. Rearranged parts of an actual face (b and c) are harder to recognize.

Each vegetable by itself does not suggest a face. Only through the arrangement of the vegetables is the face clearly recognizable. On the other hand, a rearrangement of actual face parts as shown in 1.1c is harder to recognize as a face.

It is therefore important to have methods that can recognize similar arrangements. This type of comparison between object relations, as opposed to object properties, is an analogy. As put by the philosopher Michel Foucault [33]:

> *"...the similitudes of which [an analogy] treats are not the visible, substantial ones between things themselves; they need only be the more subtle resemblances of relations."*

In all the similarity definitions proposed in this thesis, we focus on finding geometric analogies rather than similarities of object properties.

1.2 Challenges

The difficulty in finding analogies between arrangements can be understood when asking why two given arrangements are analogous. There will usually be reasons for and against an analogy and these reasons will most likely vary widely for different cases. Additionally, finding all analogies in complex scenes is a difficult task, even for a human observer. There are two main reasons why finding analogies is difficult:

Analogy Definition The concept of geometric analogies is a very broad one. A large range of similarity measures can be defined on object relations, resulting in a variety of geometric analogies. In any given modeling domain, there is usually some kind of expectation from the users

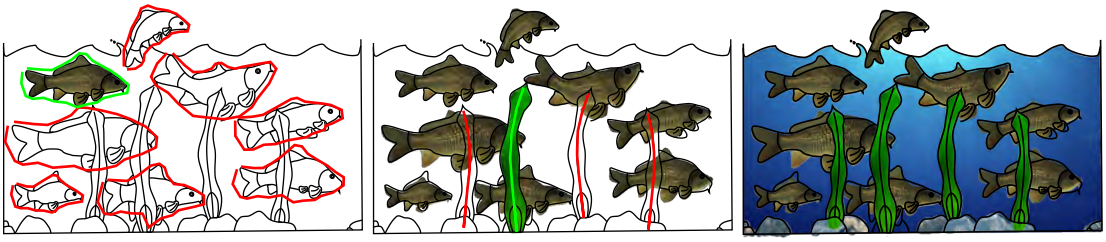


Figure 1.2: Propagating a texture between deformed fishes using our method. Given the set of contour segments (black) and a query region (green) shown on the left, our method finds similar regions (red) and establishes a dense mapping between the regions, allowing us to transfer textures, as shown in the center. In a second step, the seaweed texture is transferred as well and a background is added manually, giving the textured scene shown on the right.

as to which arrangements are more analogous than others. The difficulty then lies in formalizing this expectation, that is, finding a similarity measure that results in intuitive and useful analogies for the given domain.

Analogy Search In addition to giving intuitive and useful results, the evaluation of the similarity measure has to be efficient. Since the number of pairwise comparisons between relations is quartic in the number of elements in the arrangements, brute-force methods are usually not feasible. Methods need to be carefully designed to compare only the most relevant relations.

1.3 Dissertation Thesis

In this work we argue that meaningful similarities between shapes or shape arrangements in a scene can be found by comparing the geometric relations between the parts of each shape or between the individual elements of each arrangement and that these *analogies* enable new modeling operations that cannot be achieved with current methods. Specifically, we show that analogies between shape part arrangements can be defined that describe similarities between deformable objects, as well as analogies that describe similarities between shape arrangements that need not be related by any kind of smooth transformation, enabling edit propagation for deformable shapes and shape arrangements.

1.4 Contributions

We present two new types of geometric analogies, which we analyzed and tested thoroughly and which have interesting applications in the field of edit propagation, where propagated edits are geometrically analogous to an example edit. Additionally, we show that using multiple example edits improves the quality of propagated edits.

Geometric Analogies for Deformable 2D Shapes First, in Chapter 3, we present a method that addresses the problem of non-rigid, partial shape matching in vector graphics, based on

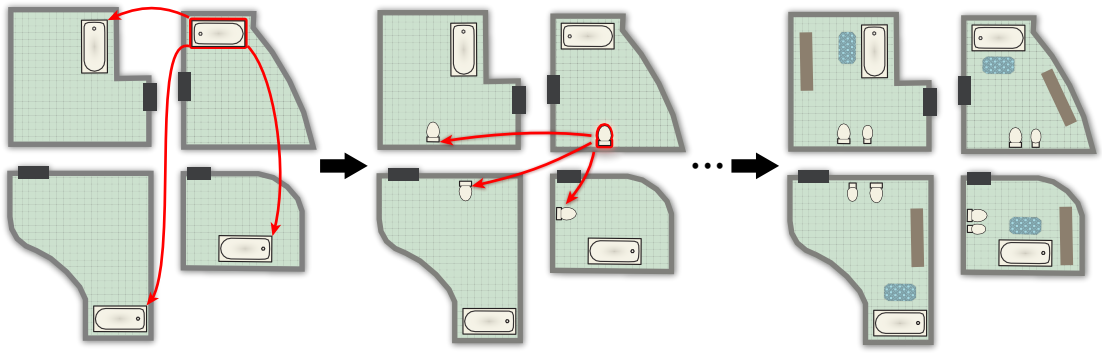


Figure 1.3: Propagating furniture placements to rooms of different shapes. The bathroom furniture placed by the user (red) is propagated to the other bathrooms using our method. After a few operations, all bathrooms are populated with a full set of furniture, as shown on the right.

analogies between shape parts. In this work, non-rigid matches are types of similarities frequently found in 'soft' deformable objects such as animals and plants, which can bend and deform and which always slightly deviate from perfect copies of one another. Given a user-specified query region in a 2D shape, similar regions are found, even if they are non-linearly distorted. Furthermore, a non-linear mapping is established between the query regions and these matches, which allows the automatic propagation of edit operations such as texturing. We propose a new definition of similarity based on the differential deviation from rigidity of matched shape regions. This type of similarity includes shapes that are intuitively similar but cannot be found by previous methods. In Figure 1.2 we show the results of transferring the texture of the fish encircled in green to the fishes encircled in red. The green region is given by the user, the red regions and the mapping between green and red regions is found by our method. Results of this work have been published in:

- Paul Guerrero, Thomas Auzinger, Michael Wimmer, Stefan Jeschke. Partial Shape Matching using Transformation Parameter Similarity. *Computer Graphics Forum. To be published.*

Single-Example Geometric Analogies for Shape Arrangements In Chapter 4, we propose a method to propagate edit operations in scenes consisting of polygons, based on finding geometric analogies between positions in polygon arrangements. In contrast to the previous method, the polygon arrangements need not be related by a smooth transformation. For example, the method allows populating large two-dimensional floor plans using relatively few furniture placement operations that are propagated to all rooms of the floor plan, even if the rooms have different shapes and different existing furniture arrangements, as shown in Figure 1.3. The basis of our approach are the geometric relations between a pose, for example the position and orientation of a piece of furniture, and a polygon arrangement, which may for example represent the shapes of rooms and furniture in a floor plan. Given a pose and the arrangement, our method finds poses

with similar relations in other polygon arrangements, even if the arrangements themselves are *dissimilar*, that is, are not related by any kind of smooth transformation. Geometric relations between pose and polygon arrangement are represented as a combination of multiple elementary relations, like the distance to a polygon boundary, or the direction to a polygon corner. Two poses are considered similar if they have many common elementary relations to their local polygon arrangements. To our knowledge, no previous method explicitly attempts to find similar poses in dissimlar polygon arrangements. Results have been published in:

- Paul Guerrero, Stefan Jeschke, Michael Wimmer, Peter Wonka. Edit Propagation using Geometric Relationship Functions. *ACM Trans. Graph.* 33, 2, Article 15, 2014

Multi-Example Geometric Analogies for Shape Arrangements In some cases, providing a single example operation is not enough to uniquely determine the intention of the user. For example, if the user places a chair between a wall and a desk, it is not clear if the relation to the wall or the desk is more important. To disambiguate the user intent and get better results, we propose a method to learn the importance of each elementary relation from multiple example poses provided by the user. This allows us to handle poses with complex relations more reliably. Additional interesting applications become possible when using axis-aligned rectangles as a base for the operations instead of poses. Since we are working with analogies, and only the *relations* between rectangles and polygon arrangements are compared for similarity, not *properties* of the rectangles, the rectangles themselves are allowed to change width and height to a configuration that fits best into the surrounding polygon arrangement. In an extension of the pose-based method, we show that using rectangles allows the propagation of more varied objects, which adapt their dimensions to the surrounding geometry such as windows, flower boxes and doors on facades. Finally, extensions are introduced that enable editing and propagation of shapes in 3D scenes. These extensions allow us to generate complex three-dimensional scenes such as large and varied city areas or large and detailed skyscrapers with thousands of objects using relatively little user interaction. This method is described in Chapter 5 and will be published in:

- Paul Guerrero, Stefan Jeschke, Michael Wimmer, Peter Wonka. Shape Placement by Example using Geometric Relationship Functions. *To be submitted*.

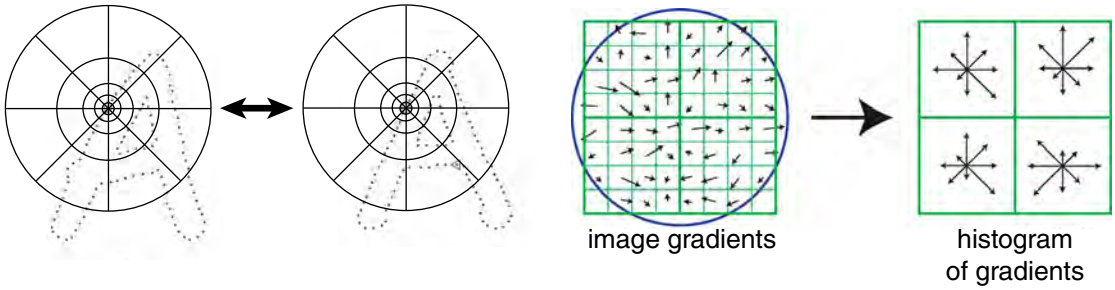
CHAPTER 2

State of the Art

In our work, we present novel geometric analogies that enable edit propagation in new modeling domains. This touches two main research fields: shape similarities and shape modeling. At first glance, the two fields may not seem related and traditionally there has been little connection between them. However, the high-level representations employed by many recent shape modeling approaches rely heavily on shape similarities. When examining the relation between shape similarities and shape modeling more closely, this does not come as a surprise: to create intuitive modeling tools, high-level shape representation must be used that capture the way people understand and reason about shapes. Since similarities are an integral part of human shape understanding, they are a necessary component of high-level shape representations. Large bodies of work exist in both fields and in this chapter, we provide an overview of methods related to our work.

2.1 Shape Similarities

Geometric analogies are special kinds of shape similarities that are based on relations between shapes or shape parts rather than properties of the shapes themselves. In the following overview, we concentrate mostly on non-rigid similarity measures, since the similarity measures presented in this thesis are non-rigid as well. Here, we understand *non-rigid* in the loose sense that is often used in literature: similarities between shapes related by complex transformations having potentially many degrees of freedom and where the exact number of degrees may depend on the input data. For a more complete overview of shape similarities, we recommend the surveys of Kaick et al. [94] and Mitra et al. [74]. The input domain of the methods presented in this section include raster images, 2D and 3D shapes, contour segments and structured point clouds. Since all of these inputs are assumed to represent shapes, we will subsume them under the term *shape*.



(a) Shape Contexts (b) SIFT

Figure 2.1: Two popular point descriptors. Shape contexts [7] (a) are radial histograms of points, SIFT descriptors [63] (b) are histograms of gradients. (Illustrations adapted from Belongie et al. [7] and Lowe et al. [63].)

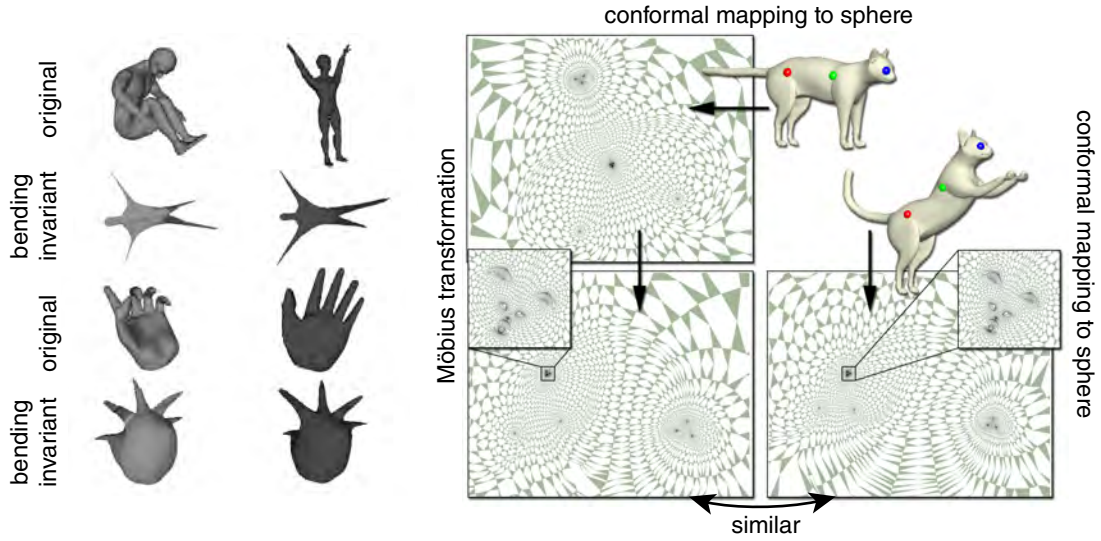
Point Descriptors

Point descriptors describe the properties of a point in a shape, usually with a vector of values. The similarity between points can then be defined as some distance measure between the vectors. The size of the neighborhood around the point that is considered to compute the properties varies between descriptors. Some descriptors consider the whole shape, some only describe infinitesimal properties of the point. Point descriptors are mainly used to establish point correspondences between shapes, that is to match points on the shapes that have similar point descriptors. These correspondences are often the first step to establish a consistent mapping between shapes or shape regions.

A well-known point descriptor for 2D shapes is called shape context [7]. This descriptor characterizes the boundary of a shape around a given point using a radial histogram over angle and distance from the point, as shown in Figure 2.1a. The boundary is discretized as a point set and the radial histogram captures the distribution of the point set around the central point. Vectors of histogram bin values can be compared using the χ^2 test statistic. Shape contexts are not intrinsically invariant to rotation or scaling of the shape, but the authors provide simple methods to make the descriptors scale and rotation invariant.

SIFT descriptors [63] apply a similar concept to images. Since there are no explicit boundaries in images, a histogram of gradients around the given point is used instead of the boundary histogram. The resulting histogram is three dimensional: it discretises the 2D position and gradient orientation of pixels relative to the central point (see Figure 2.1b). Similar to shape contexts, this descriptor is not intrinsically invariant to scale and rotation, but the authors suggest using a pyramid of Gaussians to achieve scale invariance and statistics over the gradient orientations for rotation invariance.

Both shape contexts and SIFT descriptors describe local properties of a point. The heat kernel signature [92, 40] on the other hand, is a global point descriptor. It computes a function called *signature* for a given point on a three-dimensional shape that depends on the geometry of the entire shape and the position of the point on the shape. The signature is based on the concept of heat diffusion over a surface, which depends on the shape of the surface. At a given point, it is defined as evolution of temperature over time when starting with unit heat at the point and zero heat



(a) Bending Invariant Signatures (b) Möbius Voting

Figure 2.2: Two well-known methods based on shape isometries. Bending Invariant Shape Signatures [24, 25] (a) are isometric representations of a shape, where the euclidean distance approximates the geodesic distance. Möbius Voting [59] (b) is a method to find an isometric mapping between two genus zero shapes, where the isometry is modeled with only six degrees of freedom as rotations on a unit sphere . (Illustrations adapted from Elad and Kimmel. [25] and Lipman and Funkhouser. [59].)

everywhere else. This point descriptor is intrinsically invariant under isometric transformations and has the desirable property that the signature at small time values only describes the local neighborhood of a point and becomes increasingly global over time.

Isometry-Based Similarities for 3D Shapes

A popular and useful class of similarity measures for non-rigid three-dimensional shapes are based on isometries. By considering the observation that geodesic distances on the surface of non-rigid objects such as humans or animals are generally invariant, similarity is defined as the preservation of geodesic distances between point pairs on a surface. Several methods have been proposed that use this approach to similarity and in the following we will discuss the most relevant examples.

Isometries for non-rigid shape matching were made popular by the work of Elad and Kimmel [24, 25], where 3D shapes are transformed to an isometric representation called bending invariant surface or isometric signature, where the euclidean distances approximate the geodesic distances of the original shape as closely as possible. See Figure 2.2a for a few examples of signature shapes. These signatures can then be compared instead of the original shapes using traditional rigid-matching techniques. The signatures are found using Multidimensional Scaling [55].

Mémoli and Sapiro [75] formalize this approach applied to point clouds using the Gromov-Hausdorff distance [41], which is defined as the smallest Hausdorff distance over all isometric

embeddings of two metric spaces. In this case the metric spaces are two point clouds originating from densely sampled surfaces with a metric approximating the geodesic distance on the surfaces.

Bronstein et al. [15] improve the original approach of Elad and Kimmel using a generalization of Multidimensional Scaling that also allows partial surface matching, but is less efficient than the original, since geodesic distances have to be computed between pairs of surface points.

Lipman and Funkhouser [59] note that isometric mappings between surfaces of genus zero can be broken down into a composition of two conformal- and one Möbius mapping, which only has six degrees of freedom in total. This allows to fully specify an isometric mapping using only three point correspondences. More specifically, the authors show that isometric mappings between two shapes, A and B, are equivalent to the composition of a conformal mapping of A to the unit sphere, followed by a Möbius transformation from the unit sphere to itself and a conformal mapping of the unit sphere to B (see Figure 2.2b). This composition can be uniquely defined with three point correspondences, making the space of all possible isometries low-dimensional enough to find a good solution using simple random sampling. For each isometry, a deformation error is computed measuring the quality of the mapping.

While isometry-based similarities provide excellent results for a large range of non-rigid objects, there are also many types of similar non-rigid objects that are not related by isometries. Generally, isometries are not usable if the surface of an object can stretch and contract by a significant amount relative to the size of the object. For example, the bending motions of a fish as it swims through the water contract and expand the skin of the fish, precluding the use of isometries. Additionally, isometries are not usable on sets of disconnected objects, since they can only be computed between points that are connected by a surface.

Transformation-Based Similarities

A different class of methods base their similarity on the parameters of the transformations that relate small parts or points on two shapes. Similar shape regions then consist of parts or points that have similar transformations. For example, if the heads of two persons are related by a rigid transformation, all the correctly matched parts on the two heads share the same transformation. This approach can be used for some types of non-rigid similarities as well. Take for example the different poses of a fish as it swims through the water. These poses are related by a non-rigid transformation: not all parts of the fish, such as the head or the fins, are transformed in the same way. But locally, for neighboring parts of the fish, the transformations are similar.

3D Shapes Gal et al. [35] first compute a set of local shape descriptors for each shape and then identify groups of shape descriptors that form salient shape regions, usually regions that contain interesting geometry, such as strong curvature, local minima and maxima of curvature and large curvature variance. When comparing two salient regions, the largest set of shape descriptor correspondences that agree on their transformation define the partial match and the corresponding rigid transformation between the two salient regions.

In a similar approach, Mitra et al. [73], compute partial similarities between two shapes by identifying sets of matching point descriptors with similar affine transformation parameters. The surfaces of both shapes are first covered with a set of sample points, which are then paired based

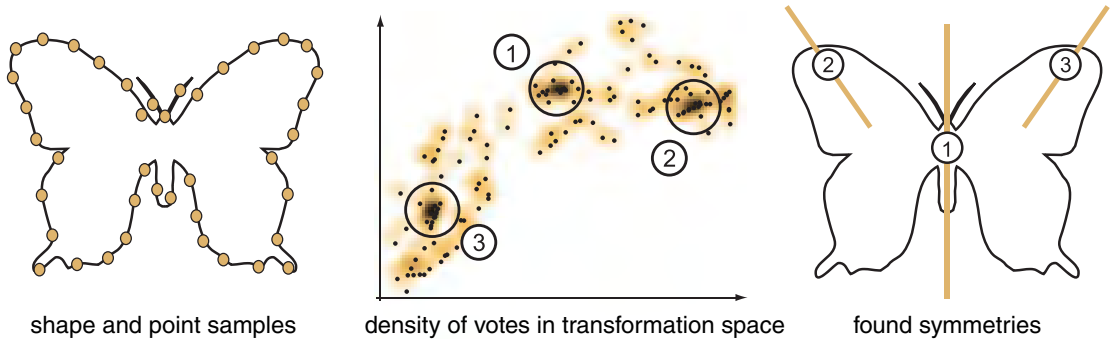


Figure 2.3: The symmetry detection method of Mitra et al. [73]. All pairs of shape sample points (left) vote for the transformation that aligns the pair in the space of transformation parameters (center). The dominant modes of the vote density correspond to dominant symmetries (right). The method also works on 3D shapes. (Illustrations adapted from Mitra et al. [73])

on similar principal curvature magnitudes. Each pair votes for the transformation relating the two point descriptors. The modes of the resulting distribution correspond to sets of matched point descriptors having the same affine transformation and define partial similarities of the two shape. See Figure 2.3 for an illustration of the individual steps.

In subsequent work, Mitra et al. [72] modify a given shape to maximize the self-similarity under affine transformations, while staying as close to the original as possible. Similar to their previous method, matched shape descriptors vote for the transformation between them. Different to previous work, their shapes start out not having affine self-similarities, which means that the votes do not form compact clusters in the space of transformations. The shapes are then optimized so that the votes of descriptor pairs form clusters that are as compact as possible, resulting in shapes that are as self-similar as possible under affine transformations.

Raster Images Ferrari et al. [26] employ a transformation voting scheme in the context of a raster-image object detection framework. They first extract a model of the desired object shape from a few lightly annotated example images. Then they find matching shapes in new images by letting matched parts of the shape model vote for the translation and scale needed to align the matches. The resulting maxima correspond to rigid transformations of the shape model that align as many of the matched shape parts as possible. To handle non-rigid transformations, the authors use the rigid transformations as initializations for the TPS-RPM method [20] (discussed in the next section) that can align two non-rigidly transformed point sets given good initial alignments. The examples shapes given by the user as well as the matched shapes all have to be related by approximately rigid transformations (plus uniform scaling). Strong deviations from rigidity, such as bent objects, can not be handled.

Similar to Ferrari et al., Lu et al. [64] use transformation voting in a object detection framework for raster images. Here, the shape model is defined manually and a a sequential Monte-Carlo method is used to find matching shapes in new images. Shape parts of the model are matched sequentially to contours in the new image and a new match is only added to the sequence if it fits

the transformation agreed upon by the model parts that have already been matched. As in Ferrari et al., only approximately rigid shapes can be matched.

The well-known Patch-Match method [4, 5] is used to find similar regions in raster images. Low-level matches of small image patches are established using photometric, rather than geometric information. The best matches are used as seeds to grow larger image regions consisting of matched patches with similar transformations as the seed match. Since all patches in matched image regions have to share the same transformation, regions related by a non-rigid deformation can not be found.

Deformation-Based Similarities

Several methods use the deformation energy needed to align two shapes as a measure of similarity. The energy can be defined in several ways, but is usually based on the intuition that physically aligning similar objects requires less deformation than aligning dissimilar objects. The isometry-based and transformation-based similarities discussed in the last sections can be understood as special cases of deformation-based similarities. For isometries, the deformation energy is defined as the amount of surface expansion or contraction needed to align two shapes. For transformation-based similarities the deformation energy is based on the change of the transformation that aligns local parts of the two shapes.

3D Shapes Zhang et al. [103] define an energy function based on mean-value encoding [52] for aligning matched points of two shapes. Shapes are sampled with a very sparse set of points, typically 8-12 points for a shape of medium complexity, such as an animal or a human. The point correspondence with minimum deformation energy is found using a search tree in the exponential space of point correspondences and for each candidate correspondence a deformation has to be computed that aligns the corresponding points. This step is relatively expensive, limiting the number of sample points. Since the set of points is sparse, only the coarse structure of two shapes can be matched using this method, a dense correspondence can not be established. Therefore it is best used on shapes that have the same global structure (e.g. same number of extremities), but are dissimilar otherwise.

Chang et al. [16] match two articulated shapes by first partitioning them into a small sets of subregions that are related by rigid transformations, such as the upper and lower arms, upper and lower legs, heads and torsos of two humans. Typically there are multiple candidate matches for each rigid subregion, for example one arm of the first person might match either arm of the second person. To get a consistent matching between the subregions, an energy term is introduced that penalizes the change in edge lengths between neighboring vertices on the shape. This method can only be used on articulated shapes that consist of rigid subparts. It cannot be used on shapes related by a smooth non-rigid transformation, such as bending.

2D Point Sets One line of research [20, 104, 66] aims at finding the transformation that optimally aligns two given sets of 2D points by simultaneously solving for optimal point correspondence and minimum deformation energy. The point sets are usually sampled from some

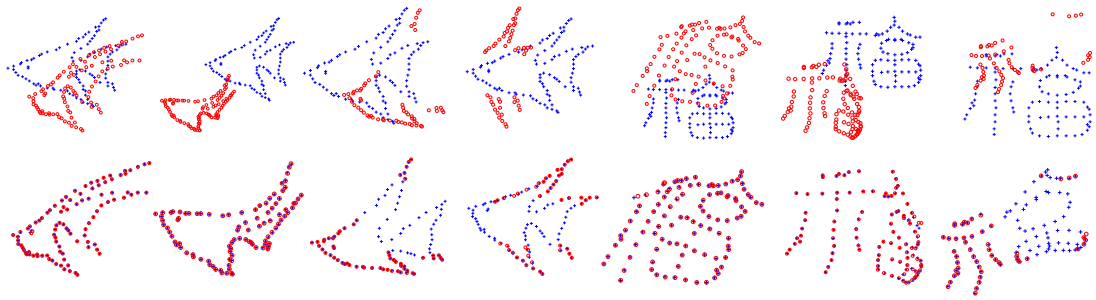


Figure 2.4: Deformed point sets (top row) aligned with the method of Ma et al. [66] (bottom row). Point set deformations are modeled as lying in a specific reproducing kernel Hilbert space and the parameters of this deformation are found using a robust L_2 estimator. The resulting method can match strongly deformed point sets with noise and missing data, but can only reliable handle a single matching subset of points. (Illustrations adapted from Ma et al. [66])

kinds of shapes and the total deformation energy of the optimal alignment can be understood as the similarity of the two shapes.

Chui and Rangarajan [20] use a thin-plate spline model for the deformation energy and a soft-assign [83] method for point correspondences. Deterministic annealing is used to alternatingly optimize correspondence and transformation of the point set. The point correspondence is relaxed to be fuzzy using the soft-assign method, allowing for efficient optimization, and the fuzziness is gradually decreased as the annealing temperature is reduced until final convergence.

In later work, Zheng and Doermann [104] use a deformation energy that penalizes changes in the neighborhood structure of the point clouds. More specifically, deformation energy only needs to be applied for points that were in the neighborhood of a point in the original point set but are no longer in the neighborhood of the same point in the deformed point set. To define a neighborhood, the authors start with a fully connected point set and remove the longest edges until the average number of neighbors in the point set is a constant (5 in their examples). Relaxation labeling [85] is employed to update the correspondences between the point sets in an iterative process that minimizes the deformation energy.

More recently, Ma et al. [66] model the deformation as a function that is restricted to lie in a specific reproducing kernel Hilbert space. They provide an L_2 estimator for the deformation that is robust to outliers. Point correspondences are found using the Hungarian method [56] with correspondence weights based on comparing shape contexts [7] centered at each point. To find the optimal alignment between two point sets, they alternate between optimizing the deformation and the point correspondence. Results of the method are shown in Figure 2.4.

The three methods to align point sets discussed above [20, 104, 66] do not require connectivity information, which is often an advantage if the full geometry is not available. They work well when aligning two sets of points representing deformed versions of the same shape, even if the point sets contain noise or if some parts of the shapes are missing. However, they do not work well if the shapes are not the dominant element of the scene, i.e. if there are several other shapes in the point sets. In this case there might not be a single best match for each part of the point set and to achieve convergence to the desired alignment, the methods need a good initialization that

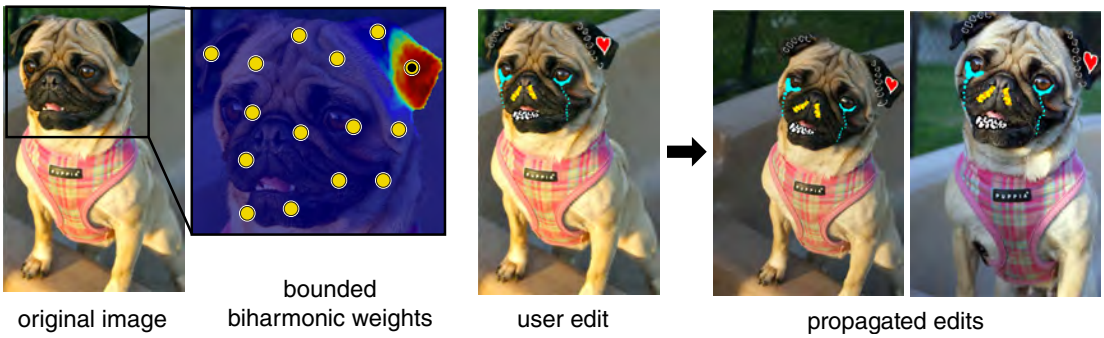


Figure 2.5: The method of Yücer et al. [102] for propagating edit operations between photographs taken from different viewpoints. A region of interest in an image is segmented into a set of soft overlapping segments, defined by custom weight functions. User edits are then transferred to similar regions in different images by assigning an affine transformation to each image segment that matches it to the corresponding segment in the new image. A weighted combination of these affine transformations is used to transfer each pixel of the user edits. (Illustrations adapted from Yücer et al. [102], photograph courtesey of Jason Jenkins.)

is already close to the desired solution. Otherwise the methods are prone to get stuck in a local minimum where different parts of a single shape in one point set are aligned to several different shapes in the other point set.

Raster Images Leordeanu et al. [58] present a method with low computational complexity to find similar regions in raster images for any given similarity metric. Point assignments are assumed to be given in the images and large clusters of consistent point assignments are found using a spectral method on the graph consisting of point assignments as nodes and the similarity between point assignments as edge weights. The principal eigenvector of the weighted adjacency matrix is used to find sets of point assignments with high pairwise similarity, corresponding to similar image regions. The authors demonstrate the method using similarities based on thin-plate spline deformation energy and a similarity measure based on the preservation of pair-wise point distances.

Srinivasan and Shi [91] use deformation-based shape matching in a raster image object detection pipeline. Shapes are represented as sets of noisy and possibly incomplete image contours. After learning a model shape from multiple lightly annotated examples, similar shapes are found in new images based on the similarity of their contours to the model contours and penalized based on the deviation of their contour arrangements from a simple translation of the model contours. Penalization is weighted separately for each model contour with weights learned from the variations in the training images, allowing for slight non-rigidity of the matched shapes.

More recently, Ma and Latecki [67] proposed another method using deformation-based similarity for object detection in raster images. The similarity is based on how well the distance and angle between pairs of points on the model contours are preserved in the matched shape. As in the method of Srinivasan and Shi, this allows for only slight non-rigidity in the matched shapes.

As part of a framework to transfer edit operations between photographs of objects from different viewpoints and in various lighting conditions, Yücer et al. [102] define similar image regions as regions that are related by linear combinations of affine transformations, where the influence of each affine transformation is modeled with novel content-aware bounded biharmonic weight functions [45] (see Figure 2.5). The weight functions are made content-aware by embedding them in the higher-dimensional space of image position *and* color, so that they decay more strongly at image contours. This results in interesting transformations that are more rigid in smooth image regions than at contours. The parameters for the individual affine transformations that optimally align two image regions are found using the Lucas-Kanade framework [65, 3] with an initialization based on SIFT point descriptors [63]. However, only a single similar region can be found per image.

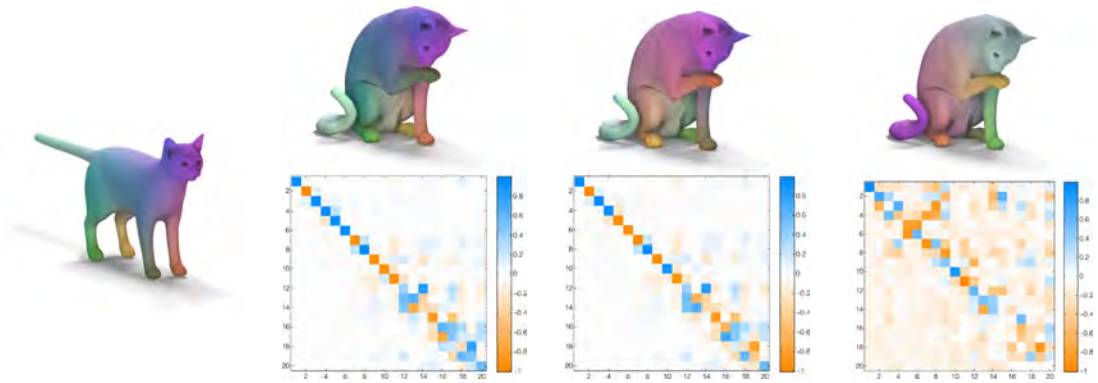
The projection error is a well-known measure of the difference between two mappings or transformations, based on the distance between points transformed with one or the other mapping. It is used in literature to measure deformation energy as the projection error between the transformations of neighboring point correspondences. The most relevant methods using the projection error include a framework for automatic image enhancement such as color or lighting correction using as template a correctly colored and lit image containing similar objects [43] and a method for region matching given local point correspondences [19].

The first method [43] establishes point correspondences between the pixels of two images using the Patch-Match method [5] described earlier. Nearby correspondences are then aggregated to consistent regions that have low projection error. However, the Patch-Match method only finds a single best match for each pixel, causing problems when there are multiple repeated regions in an image, such as repeating patterns or several similar objects.

The second method [19] uses SIFT features [63] to establish point correspondences between image features. A distance based on the projection error is defined between correspondences and used to find clusters of correspondences with low projection error between neighbors using hierarchical linkage clustering. However, since the projection error increases or decreases with the density of feature correspondences in an image, this method is less suited for images with strongly varying feature density.

Other Similarity Types

Several other interesting non-rigid similarity types have been presented that do not fit into any of the categories given above. Berner et al. [8] automatically infer a similarity type from a given set of example shapes that have a common connectivity structure of salient features such as corners or edges. They identify a low-dimensional subspace that contains all the example shapes in the full space of all shapes having the given connectivity structure. The subspace is represented as a linear combination of basis shapes, which are found using Principal Component Analysis on the example shapes. Similarity can then be defined as the distance of a shape to this subspace. The main advantage of this method is that the main modes of variation of the non-rigid shape similarity can be found automatically given a scene containing elements with a similar connectivity structure of salient features. However, it does not work on shapes that do not have a similar connectivity structure or the salient features and their connectivity is not clear.



(a) source (b) ground-truth map (c) left to right map (d) head to tail map

Figure 2.6: Correspondence between cat models in different poses represented by the method of Ovsjanikov et al. [77]. Correspondences are modeled as a linear transformation on the coefficients of Laplace-Beltrami basis functions defined on the surface of the shapes, shown as matrices in the bottom row. A nearly isometric correspondence is represented as a nearly diagonal matrix (b and c). Non-isometric correspondences are represented by non-diagonal matrices (d). (Illustrations adapted from Ovsjanikov et al. [77].)

Additionally, quite a few examples are needed to properly span the subspace, but may not always be available.

As part of a larger framework of interactive repetition finding, completion and manipulation in raster images, Cheng et al. [18] propose a similarity based on the agreement of contour positions and orientations between a template region coarsely marked by the user and the remaining image. Matches for the template are found in a brute-force approach by moving the template to each image pixel and computing the contour agreement. Additionally, a small set of rotations and scalings of the template is tested at each pixel. As a result, only small rotation and scale differences can be found efficiently. Slightly non-rigid deformations can be handled by dilating the contours in the image.

Ovsjanikov et al. [77] propose to establish a correspondence between functions rather than points on the two shapes. They call this type of mapping *functional maps* and it has a number of advantages over point-wise mapping. Functions on both shapes are represented in a multi-scale functional basis such as the Laplace-Beltrami eigenfunctions. A correspondence between the shapes is established with a matrix that transforms basis coefficients of one shape into coefficients on the other shape. An example is shown in Figure 2.6. This coefficient transformation maps the function on one shape to the corresponding function on the second shape and does not depend on the choice of function. Due to the matrix formulation, it is possible to use linear operations on the mapping that can not be used on a point-wise mapping. To define concrete mappings between two shapes, various constraints on the correspondence matrix can be defined and mixed, such as point correspondences or functional preservation, where custom functions are preserved by the mapping. Note that properties of the shape such as curvature can be encoded as function preservation constraints by creating a function that has as value the desired property, ensuring for example that curvature is preserved by the mapping.

Pokrass et al. [81] use functional maps with pairs of corresponding salient regions on both shapes as constraints. Neither the point correspondence inside the regions, nor the correspondence between the salient regions is known in advance. The authors provide a method to solve for both correspondences simultaneously using permuted sparse coding. However, the method relies on nearly identical Laplace-Beltrami basis functions on both shapes, which is only the case for shapes that are nearly isometric.

Generalized Coordinate Systems

While coordinate systems are not explicitly designed to find shape similarities, they can be used to identify similar points in two shapes by defining the similarity as some distance in the coordinate space. Several coordinate systems for points inside polygons or polyhedra have been presented. These polygons or polyhedra are usually employed as a control cage to deform a shape defined inside the polygon/polyhedron. The coordinates thereby establish a dense correspondence between points inside the original and deformed polygons/polyhedra, given a sparse correspondence between their vertices.

Mean-value coordinates [31, 32, 48, 44, 57], are a generalization of barycentric coordinates to arbitrary planar polygons or polyhedra that retain a number of desirable properties such as partition of unity, smoothness and invariance to similarity transformations. Similar to barycentric coordinates, mean-value coordinates can be computed with a simple closed-form expression.

Harmonic coordinates [47] are an alternative to mean-value coordinates that is better suited for character articulation. Similar to mean-value coordinates, harmonic coordinates also have as dimension the number of polygon or polyhedron vertices. Their main advantage that allows for more intuitive character animation is that they are always positive and are based on the distance of a point to a vertex *within* the polygon or polyhedron, as opposed to the straight-line distance. They are computed as solutions to Laplace's equations inside the polygon or polyhedron and as a consequence, they do generally not have a closed-form solution and must be approximated using a numerical solver, as opposed to the simple closed-form expression of mean-value coordinates.

Green coordinates [60] relax the adherence of the shape to the control cage in favor of a conformal or quasi-conformal (i.e. more shape-preserving) mapping between the original and deformed shapes. Unlike the previously described coordinates, the dimensionality of green coordinates corresponds to the number of polygon or polyhedron vertices plus the number of control edges (2D) or faces (3D). Similar to mean-value coordinates, they have closed-form expressions, although the expressions are less simple and elegant.

2.2 Shape Modeling

Most current research in shape modeling that is related to our work can roughly be divided into procedural modeling, analyze-and-edit approaches and modeling by example. Due to its lack of automation in the process of finding a high-level specification of the model, procedural modeling is less relevant to our approach and will not be discussed in the following (for an overview we refer to Smelik et al. [89]). Edit propagation can be considered a specialization of modeling by example, where examples of edit operations are provided instead of examples of the desired

results. Analyze-and-edit as well as modeling-by-example approaches have been developed in the field of 3D shape modeling and consequently their input domain is usually restricted to 3D shapes. In contrast, edit-propagation methods come from the field of image editing and most research is focused on manipulating images. However, the central ideas proposed in these methods are usually input-agnostic and could be applied to other input domains as well.

Analyze-and-Edit

Current commercial 3d editors either provide a limited set of fixed high-level shape modeling tools, such as deformations with a predefined cage, bending, deformation along a path, etc., or very low-level editing tools, such as direct vertex modifications. For many operations, the low-level tools suffer from too many degrees of freedom, making seemingly simple editing operations, such as changing the height of a house while preserving details or moving the arm of a character, tedious and time-consuming operations. High-level tools provided by the programs usually only have a very limited set of operations that do often not include the domain-specific task required by a user; in other words, the provided degrees of freedom are not adapted to the domain or the shape the user is working on.

The problem lies in the lack of high-level specification these tools have about the shape that is being modeled. People usually see shapes not as a collection of vertices and faces, but as a sets of functional parts, connected by some kinds of relations. For example, the roof sits on top of a house, arms are connected left and right to the torso. In a given domain, these relations generally have a specific structure, for example in man-made objects, parallel lines or right angles are usually preferred. It would therefore make sense for modeling tools for this domain to respect this structure relations and remove degrees of freedom that violate them. *Analyze-and-edit* approaches that encode this structure in a domain-specific set of constraints. They first analyze a model to choose relations from the set that are applicable to the model and then provide modeling tools that are constrained to respect the relations.

Single Shapes Early work in this area focuses on analyzing and preserving low-level shape characteristics, such as normals, curvature, local rigidity or local volume [13], or manually define constraints on the shape [14, 82, 69, 68].

More recently, Kraevoy et al. [53] describe a method for structure-preserving non-uniform scaling of complex shapes. Given a requested non-uniform scaling along any axis, the shapes are analyzed and parts where the scaling would result in a distortion of the structure are identified, based on slippage analysis [37] and change in normal curvature in direction of the scaling axis. To achieve the given scaling, parts that are vulnerable to distortions are not scaled and non-vulnerable parts are scaled by a larger amount instead.

An analyze-and-edit method to automatically detect joints in 3d shapes has been proposed by Xu et al. [98]. The geometry and type of joints are detected using slippage analysis [37] and intersection tests for the range of motion. In subsequent edit operations, the shape can be deformed intuitively, by constraining the components connected by joints to be either rigid or slightly elastic.

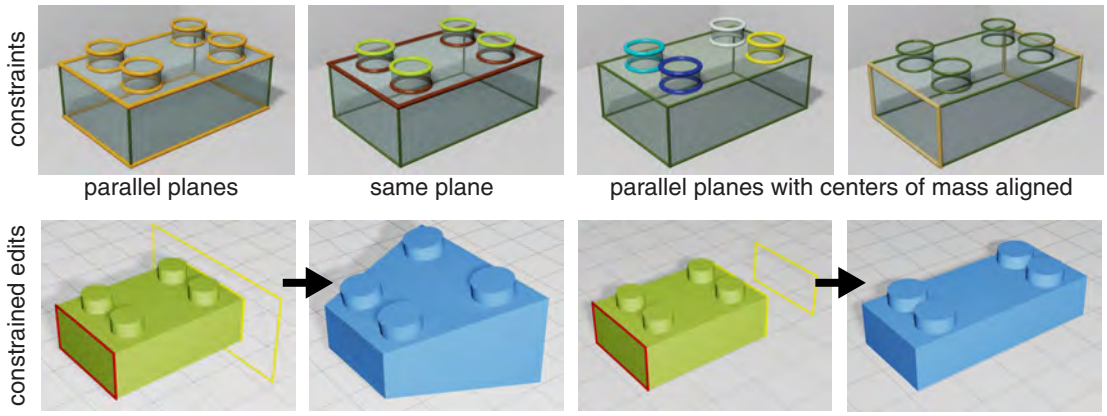


Figure 2.7: Simple edits operations as proposed in the analyze-and-edit method by Gal et al. [36]. Edit operations are constrained to respect as many relations between wires as possible. Examples of relations are shown in the top row and two simple edits and their results in the bottom row. (Illustrations adapted from Gal et al. [36].)

Gal et al. propose an analyze-and-edit method for man-made shapes called *iWires* [36]. Salient curves on the shapes called *wires* are extracted and relations between the wires, such as planarity or aligned centers of mass, or relations between parts of the same wire, such as equal wire corner angles or equal wire lengths, are found. These relations serve as constraints that have to be maintained by subsequent edit operations. For example, when editing one vertex of a shape, the other vertices are altered to follow the constraints as closely as possible. Examples are shown in Figure 2.7. The authors argue that this high-level specification based on salient curves and their relations is descriptive enough to allow for efficient and intuitive editing of a wide range of man-made objects.

Zheng et al. [106] analyze a shape to identify a hierarchy of functional parts, structure-preserving transformations on parts and relations between parts, such as symmetry, coplanarity and parallelity. The authors observe that this hierarchy of parts provides a more robust and descriptive high-level specification of the shape than the wires extracted by the iWires method [36]. Similar to iWires, edit operations of a part are constrained to be structure-preserving and each edit is propagated to other parts of the shape to maintain relations between parts. However, shape analysis is not fully automatic, user intervention is required in some cases.

In concurrent work, Bokeloh et al. [11, 12] extract a pattern-based structure specification from a shape. Continuous and discrete one- and two-dimensional translational patterns are detected in the shape geometry. Discrete patterns may, for example, be the wooden boards on a park bench or the windows on a facade. Continuous patterns refer to slippable parts of the shape, such as cylinders or planes, where the individual elements (e.g. parts of the cylinder) are related by continuous translations. Relations between different patterns, such as concurrent start- and endpoints, are identified as well. Edit operations on the shape are constrained to preserve the patterns and relations. For example, giving the bench a higher back rest adds additional wooden boards to the back rest instead of stretching the individual boards, or resizing a house automatically adds or removes windows on the facade.

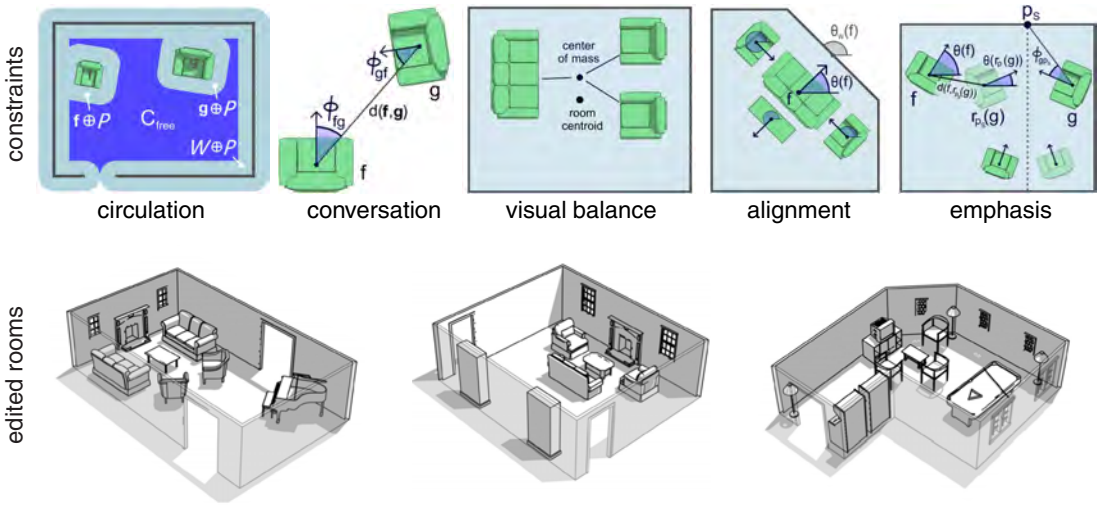


Figure 2.8: Constraints and results from the method of Merrell et al. [71]. Edit operations are constrained to respect interior design guidelines (top row), resulting in plausible furniture arrangements with relatively little user input (bottom row). (Illustrations adapted from Merrell et al. [71].)

Scenes Analyze-and-edit methods can also be employed to create scenes consisting of multiple separate shapes. Although they are not explicitly referred to as analyze-and-edit methods in literature, they share the same key idea: reducing the space of possible scenes and permissible edits by imposing constraints derived from a high-level analysis of the scene. Note that these methods, like all analyze-and-edit methods, make no attempt to learn user edits. Therefore, user edits need to be repeated for

Xu et al. [97] define constraints for interior scenes, including pseudo-physical constraints to avoid unstable configurations or interpenetration of shapes, geometric constraints such as proximity constraints that describe the permissible closeness of objects and semantic constraints that describe which type of object is usually placed on which other type of object, such as plates on a table. User operations are limited to respect these constraints as closely as possible, but certain constraints such as the semantic constraints may be overridden by user edits. Objects can also be placed automatically in permissible regions using spatial planning [38]. The constraints effectively exclude strongly inconsistent scene layouts, but do not encode enough information to create plausible scenes fully automatically. Shapes such as tables and chairs in a room still need to be placed manually using the provided editing tools. The authors demonstrate modeling scenes containing hundreds of objects in under an hour.

Merrell et al. [71] define a set of constraints derived from interior design guidelines. Given a set of furniture and a room, a plausible layout can be generated automatically. User edits are included as additional constraints and after each edit, a new list of plausible room layouts is generated and presented as suggestions to the user (see Figure 2.8). This method can be used to quickly generate furniture layouts for a given room with minimal user interaction.

Yeh et al. [100] specify custom soft constraints on arbitrary properties of shapes or relations between shape properties using a custom imperative programming language. Properties may for

example be the size or position of a shape, but also more specific attributes, such as the number of plates a table should support. The set of properties of all shapes uniquely define a scene. Users do not directly modify properties, instead properties are sampled from manually specified probability distributions. A custom Markov chain Monte Carlo algorithm is used to find multiple scenes having properties with high probability while satisfying the constraints. The method is used by the authors to generate plausible furniture layouts for a given room and given object property distributions without further user interaction.

Zheng et al. propose an analyze-and-edit method for images of man-made made environments [105]. Three-dimensional objects in the image are represented with cuboid proxies that are fitted to a given set of annotated interest regions while simultaneously solving for the camera parameters. The image can then be decomposed into the objects represented by the proxies and a background layer approximated by a ground plane. Relations between the proxies, such as placement coplanarity or repetitions are extracted as well. This high-level specification of the image content can then be used to move, resize, delete, add or replace objects in the image, while maintaining the relations between the proxies. Similar to previous work of Zheng et al. [106], the analysis is not fully automatic and requires user intervention in some cases.

Note that all the analyze-and-edit methods presented above are only based on a high-level specification of the shape or scene and not of the entire modeling operation; the operations performed by the user are not included in the specification. As a result, these methods can not adapt to user input. Methods that include the user input in their specification are presented next.

Modeling by Example

When modeling complex shapes or scenes, the high-level specifications extracted by analyze-and-edit methods are helpful, but are often not enough to make the modeling task feasible. The shapes that can be created by these methods are only constrained by the structure of the starting shape and the domain of the method, such as the domain of man-made objects. These constraints effectively exclude many undesirable shapes, but generally a user does not aim at just any man-made object that can be created from a starting shape, but rather has a more specific goal in mind, for example a house or even a house with a specific style. This *modeling intent* of the user is not represented in the high-level specification used by analyze-and-edit methods. Consequently, the specific goal of the user, i.e. the desired model, has to be reached from the starting shape manually through the provided edit operations. This is more efficient than in traditional methods, but can still be difficult for complex shapes or tedious in scenes where a lot of models have to be created.

Take as example a single complex shape, such as a sailing boat: analyze-and-edit methods might constrain each edit operation to preserve the structural details of the boat, but creating a new boat or transforming an existing boat to have the desired shape still requires resizing, rearranging and creating a lot of details. Especially for a novice user, this might be a daunting task. Another example might be a city scene: even though editing each shape in the scene is more efficient, thousands of shapes such as houses, streets, windows and doors still need to be placed and edited. Even for an expert user, manually creating all these objects is not feasible.

Modeling-by-example approaches aim at creating new and varied shapes or scenes from a set of examples. The key idea is to use high-level specifications that also include the modeling intent of the user, derived from the example shapes. Ideally, this modeling intent allows narrowing the

space of possible shapes to include only those that are desired by the user, allowing a much more efficient or even fully automated creation of new shapes. Additionally, unlike analyze-and-edit approaches, it is not necessary to pre-encode domain-specific knowledge as constraints, all needed information is learned from the examples. Modeling-by-example approaches trade direct artistic control for further automation of the modeling task, since the resulting models are based largely on the examples and not on direct user input. Note that it is possible to combine modeling by example with traditional approaches or analyze-and-edit methods, for example by first using modeling by example to create a large set of models, and then manually editing specific models for more direct control.

Single Shapes Early work in this area focuses on generating new shapes from shape parts taken from a model database. This research direction is also called component-based or assembly-based modeling. Inspired by the work of Sloan et al. [88], who propose to generate new shapes by morphing between N example shapes, Funkhouser et al. [34] describe a method to use only parts of the example shapes instead of morphing the entire shapes. The method is still largely manual, the user needs to mark parts on a shape to be replaced or draw proxy bounding boxes where new parts should be added. A shape database is then searched for parts that are geometrically similar to the marked parts or proxies, based on the sum of squared distances between the parts. Good matches can then be stitched to the shape to replace the marked parts or proxies, and geometry is automatically adjusted so the seams are not visible.

Merrell [70] aims at automating the process of creating new shapes from a set of examples by simplifying the parts that can be stitched together. Like in a LEGO® set, the examples and the new shapes must be constructed using a predefined set of parts that have a similar size. New shapes are assembled automatically from these parts under the constraints that two parts may only be adjacent in each spatial axis, if they have been observed in that configuration in one of the examples. Additionally, the probability of picking new parts is set so that the total number of occurrences of each part resembles those in the examples.

In two similar approaches Chaudhuri et al. [17] and Kalogerakis et al. [49] propose methods to automatically assemble shapes using parts taken from a database of example shapes. Unlike the method of Merrell [70] described above, parts may have arbitrary shapes and relations between parts other than only adjacencies are learned from the examples. Parts are extracted and labeled from a shape database semi-automatically in a pre-processing step using the method of Kalogerakis et al. [50]. Relations between parts in the database, such as the number of parts of a specific type in a shape, symmetries and adjacencies between parts, as well as the properties of individual parts, such as labels, curvature, diameter and several other geometric descriptors, are used to form object categories and train a probabilistic model. The model can then be used to assemble new shapes either fully automatically, or by providing suggestions for new parts during an interactive modeling session, similar to Funkhouser et al. [34]. Several results of the method are shown in Figure 2.9.

Scenes Fisher and Hanrahan. [27] apply modeling by example to interior scenes. Similar to the work of Funkhouser et al. [34], the user starts with an unfinished scene and uses a database of similar scenes to find relevant new models to place into the scene. The user draws a bounding

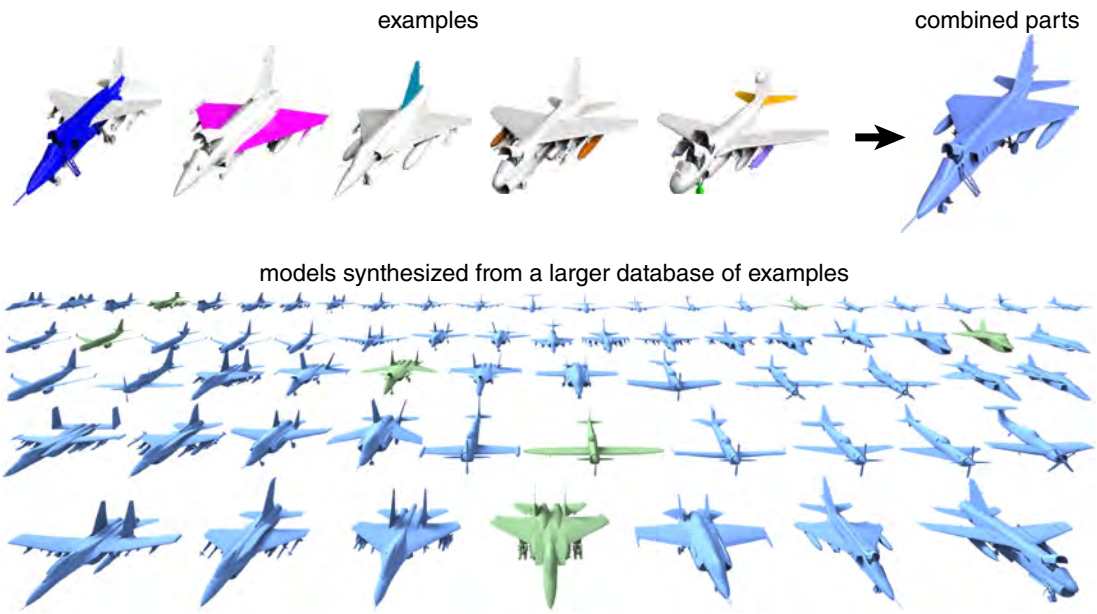


Figure 2.9: Shapes synthesized by the method of Kalogerakis et al. [49]. New and varied shapes are synthesized by combining parts taken from a database of example shapes, while respecting relationships between parts observed in the examples. The composition of a synthesized shape from individual parts is shown in the top row, a large set of synthesized shapes in the bottom row. (Illustrations adapted from Kalogerakis et al. [49].)

box as a proxy for a new object and the context of the bounding box is used to search for relevant models in the scene database, i.e. models that are surrounded by a similar arrangement of objects as the bounding box. The search returns a list of model suggestions and the user can pick a model to replace the proxy bounding box. Similarity between two contexts is based on the similarity of the distances to the surrounding objects, on the geometric similarity of the surrounding objects themselves, as well as on some additional properties of the surrounding objects if available, like tags, textures and material properties. This method can facilitate the creation of interior scenes, but the placement of new objects still needs to be specified manually, limiting its usefulness for floorplans containing many rooms.

In subsequent work, Fisher, et al. [29] improve their similarity measure for interior scenes. Scenes are described by directed graphs, where the nodes correspond to objects and the edges to relations between objects, consisting of enclosure and different types of surface contact relations. Two graphs are similar of they have similar structure, the objects corresponding to matched nodes are geometrically similar and matched edges correspond to the same relations. Similar to previous work, the method can be used for context-based model suggestions, but it is also possible to find similar scenes in a scene database.

Yu et al. [101] propose a method to synthesize realistic furniture arrangements in given empty rooms with minimal user interaction that has properties from both analyze-and-edit and modeling-by-example approaches. User interaction is only required in a pre-processing step

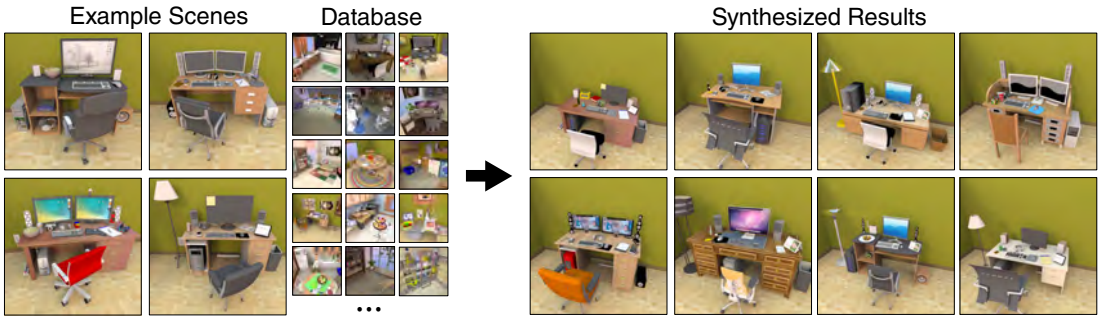


Figure 2.10: Interior scenes synthesized using the method proposed by Fisher et al. [28] from a small set of example scenes given by the user and a large database of database of generic interior scenes. (Illustrations adapted from Fisher et al. [28].)

and the automatic scene synthesis is based on predefined constraints, as well as on constraints on the relationships between pairs of furniture pieces that are learned from example scenes. Predefined constraints include accessibility and visibility of furniture (for example, a TV should not be occluded, a door should be accessible). Learned constraints include relative positions and orientations of furniture pairs, such as the relations between chair and desk. The constraints are formalized as a cost function and a minimum-cost furniture arrangement is found using a technique based on simulated annealing. The authors demonstrate generating plausible furniture arrangements for various room types from a few example arrangements. However, the pairs of furniture pieces used for pair-wise relationships need to be defined manually in the example scenes and quite a few complete furniture arrangements need to be available as examples to successfully learn furniture relationships.

Drawing on previous work in component-based modeling described above [49], Fisher et al. [28] apply a similar approach to modeling interior scenes consisting of shape arrangements, as shown in Figure 2.10. Unlike previous work on generating interior scenes, this approach is fully automatic, requiring only a small set of labeled examples that define type of scene to be generated in a modeling session and a large database of labeled generic scenes that fill in information that might be missing in the small set of examples. Object categories are automatically learned in the examples, corresponding to objects with similar context, that is, objects that are surrounded by similar objects. These categories do not necessarily correspond to the usual semantic categories like ‘apples’ or ‘cookies’ defined by the labels; a single contextual category may contain objects from several semantic categories. The examples and scenes in the database similar to the examples (using the scene similarity defined in [29]) are used to train probabilistic models that describe what object categories should be in the new scenes and where they should be placed. Using this method, a large variety of new scenes can be generated from few examples, but like in previous work, this requires fully modeling the example scenes and having available a large database of labeled generic example scenes.

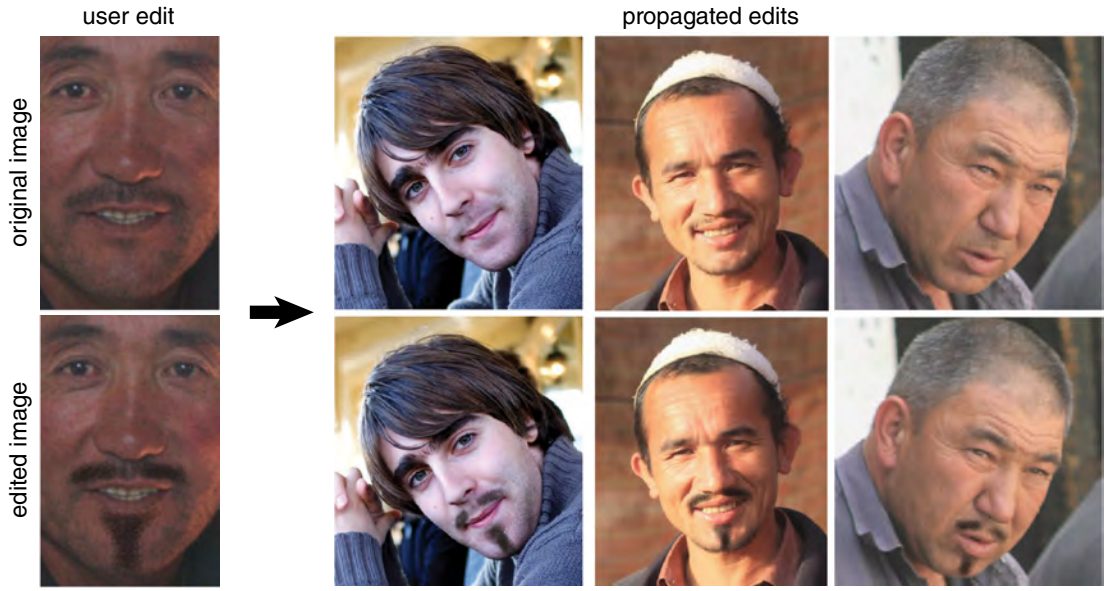


Figure 2.11: Propagating a mustache to photos of different faces using the method of Berthouzouz et al. [9]. A mustache is added to the image shown on the left and this example operation is propagated to images of different faces using a probabilistic model learned from the example operation. (Illustrations adapted from Berthouzouz et al. [9].)

Edit Propagation

Three main drawbacks of modeling-by-example approaches are the work required to create or obtain the examples, the lack of direct user control and the inherent difficulty inherent in creating a whole scene or detailed shape automatically from a small set of examples. Edit-propagation approaches fare better in the last two problems due to breaking down the synthesis task into individual operations. One or multiple examples are provided for each edit operation, which is then propagated to a potentially large number of other shapes or scenes. This gives back a measure of direct control to the user, as feedback is provided after each operation and the user might choose to undo or change an operation based on the feedback. Additionally, the propagation of individual operations is less involved than the synthesis of entire shapes or scenes from examples. As a drawback, pre-existing shape or scene databases cannot be used as examples, and the user may have to adapt the order of operations to suit the edit-propagation method. Most methods in this area come from the area of image or video editing. In this overview, we will focus on the relatively small set of methods that could be adapted to shape modeling.

Berthouzouz et al. [9] present a method to propagate edit operations between images. Two typical image editing operations are creating a selection and drawing a brush stroke. The goal of this work is to propagate these operations from a few example images to new images with similar content. Brush and selection locations are described with colorimetric features, such as the color at the brush stroke or selection, as well as geometric features, such as the xy-position relative to a landmark or a contour. A probabilistic model is learned that describes the location of

selections and brush strokes in the examples in terms of these colorimetric and geometric features and that can be applied to other images with similar content. See Figure 2.11 for the result of propagating a mustache drawing from one image of a face to several other images of faces. Using only geometric features, this method could easily be adapted to transfer edits between 2D or 3D shapes. However, compared to our work, far fewer geometric features are used and combined in a less flexible manner, limiting possible edit propagations to geometry that has roughly similar shapes.

Two other methods that can be described as edit-propagation approaches for images are the methods by HaCohen et al. [43] and Yücer et al. [102] described in Section 2.1. Their similarity measures described earlier are used to propagate tone enhancements [43] or local image edits such as brush strokes [102] from a single example to a target image. However, these methods are designed to propagate edits to a single image location, limiting their use in scenarios where edits should be propagated to multiple targets.

The methods presented in this thesis fall in the category of edit-propagation methods as well. However they have been explicitly designed to work on shapes as opposed to images and allow for greater variations between example- and target shapes.

CHAPTER 3

Geometric Analogies for Deformable 2D Shapes

3.1 Introduction

In this chapter, we present our first contribution, a method to find geometric analogies between deformed shapes, which can be used to facilitate edit operations on these shapes. Geometric manipulation of 2D or 3D shapes is a central component of many computer-graphics applications. Throughout the years, the typical shape or scene complexity has increased, and as a result, modeling tools have become increasingly sophisticated to keep up with this complexity. In chapters 1 and 2, we have presented several paradigms for intelligent modeling tools: procedural systems encode rules about viable results in a grammar, but need an expert to encode domain-specific knowledge in a typically large number of rules, which may be difficult to find. Analyze-and-edit methods constrain the modeling operations to produce only viable results, or optimize the shape to follow the constraints as closely as possible after each edit operation. Other methods provide several finished results as example, but user control is very difficult, and generated results might be of low quality if not using a large number of examples. A recent research trend is the intelligent manipulation of many similar shapes with only a single interaction. The edit-propagation approach is an example of this trend: edit operations applied to one shape are automatically transferred to similar shapes. This approach provides relatively direct user control, requires little domain-specific knowledge and only moderate modeling expertise. For a wide range of applications, this ability to quickly select and edit model parts that are geometrically *similar* – but not *identical* – can considerably increase the efficiency of the editing work flow.

This inevitably leads to the field of shape matching, which traditionally uses point correspondences as basis for more sophisticated matching techniques. A point correspondence matches a point in a user-defined query region to similar points outside (target points) according to a local descriptor, such as SIFT [63] or Shape Contexts [7]. In this chapter, we refer to these point correspondences as *first-order similarities*. In order to match a whole query region, different approaches exist. One is to find groups of point correspondences that preserve the pair-wise point

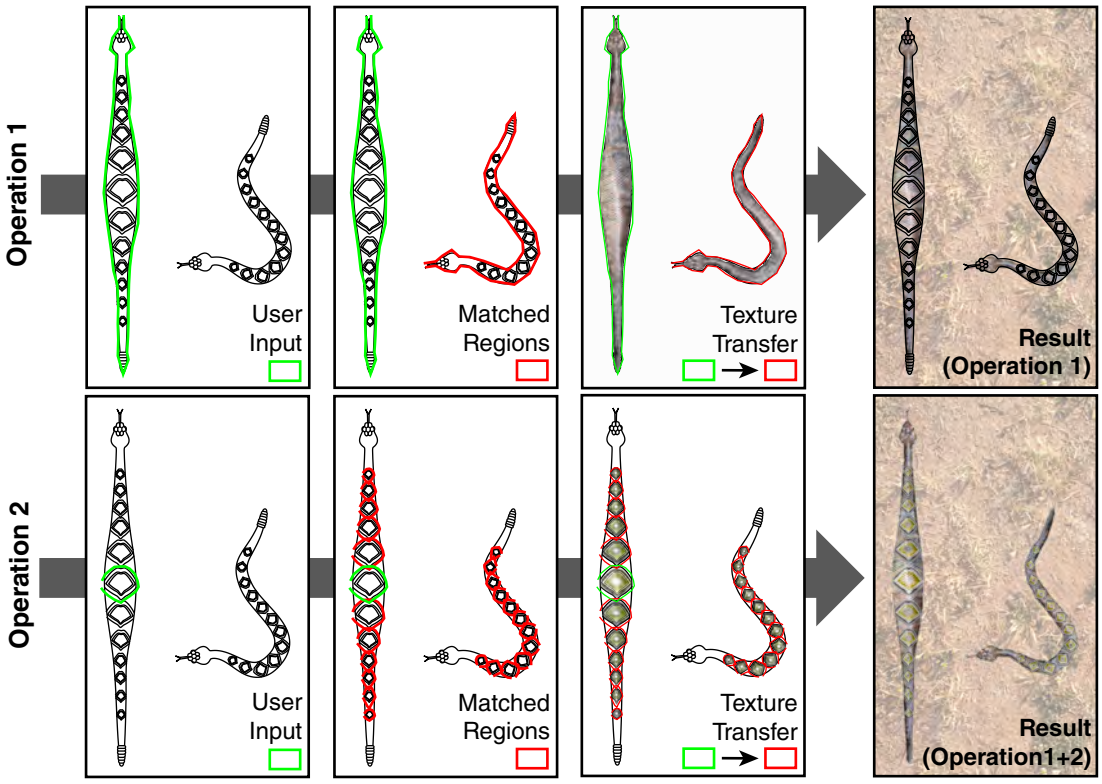


Figure 3.1: Two snakes related by a highly non-rigid transformation can be matched with our method. A user selects a region (green) and similar regions (red) are automatically found, making it possible to transfer the texture inside the green region to the red regions. Two operations are performed and the results with an added background texture are shown in the last column.

distances given in the query region in their respective target region, which is used in isometric shape matching [25, 1, 75, 15, 81]. A different approach tries to find correspondences that have *similar transformations* between query and target points [35, 73, 72] (cf. Figure 3.2a and 3.2b). The similarity between point correspondences is a geometric analogy, but to clearly separate them from first-order similarities, we will refer to this similarity between first-order correspondences in this chapter as *second-order similarity*.

Given a query region, it is often desirable to identify non-linearly distorted regions, due to perspective, object bending, different object positions and orientations, etc. Unfortunately, requiring the matched point pairs to have an identical transform only lets us find *rigid* matches. In the work presented in this chapter, we overcome this limitation by introducing the concept of *transformation parameter similarity* (TPS). We regard two regions as similar if they are related by approximately rigid first-order similarities and if the corresponding transformation parameters vary slowly over the region, i.e., have a small gradient. An example is given in Figure 3.2c, where the subregions are related by approximately rigid transforms, whereas globally, the two regions are transformed non-rigidly. Note that the rotation parameter varies slowly over the region. Our requirement of approximately rigid *local* transformations ensures that small-scale geometric detail is preserved and the region is not distorted beyond recognition.

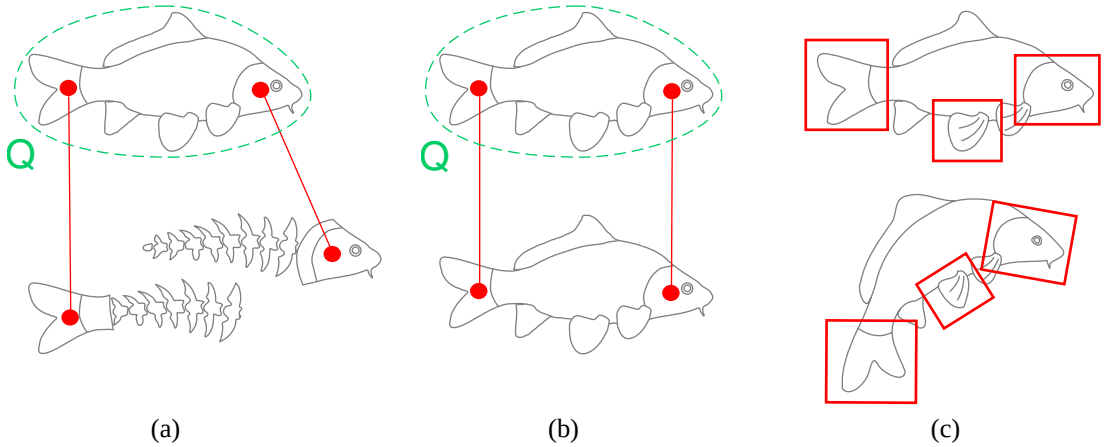


Figure 3.2: Rigidly transformed regions can be found by searching for point correspondences with similar transformations. The correspondences in (b) have a similar transformation and belong to the same rigidly transformed region, in contrast to the correspondences in (a). The two fish in (c) are related by a non-rigid transform, whereas each pair of matching subregions is transformed rigidly.

Based on this concept, the main contribution presented in this chapter is a novel method to find non-rigid matches to a user-specified query region in a vector image using the transformation parameter similarity defined above. Furthermore, a dense mapping is established between the query region and all matched regions. This allows the transfer of editing operations between complex 2D shapes, such as painted strokes or textures (cf. Figure 3.1). Currently, our algorithm focuses on 2D shapes given by contour segments, but it could be generalized to 3D shapes given by 2D boundary meshes in future work.

3.2 Transformation Parameter Similarity

Our concept of similarity is based on 'smoothly' changing transformation parameters of first-order point correspondences over a region. In the discourse of this similarity concept we introduce three central terms: a *transformation space* is the space of rigid transformation parameters, *first-order correspondences* connect two points with local neighborhoods related by a rigid transformation and a *transformation function* models the non-rigid transformation between two shape regions. In the following, we give formal definitions of these terms.

The *transformation space* is the space of rigid transformation parameters. In our method, we focus on 2D similarity transformations $S(t_x, t_y, s, r)$, parametrized by x- and y-translation t_x and t_y, uniform scaling s and rotation r, resulting in a four-dimensional transformation space with coordinates $\tau = (t_x, t_y, s, r) \in \mathbb{T}$. Since the scaling of the individual dimensions is arbitrary (e.g. it depends on using radians or degrees for the rotation), we normalize the dimensions of the transformation space. The following normalization factor is used throughout our method (unless noted otherwise): $(0.03b, 0.03b, 1.25, 45°)$, where b is the length of the bounding box diagonal of the input shape. A translation by 3% of the bounding box diagonal is equivalent to a scaling by a factor of 1.25 and a rotation by $45°$. Given parameters (t_x, t_y, s, r), the transformation of a point

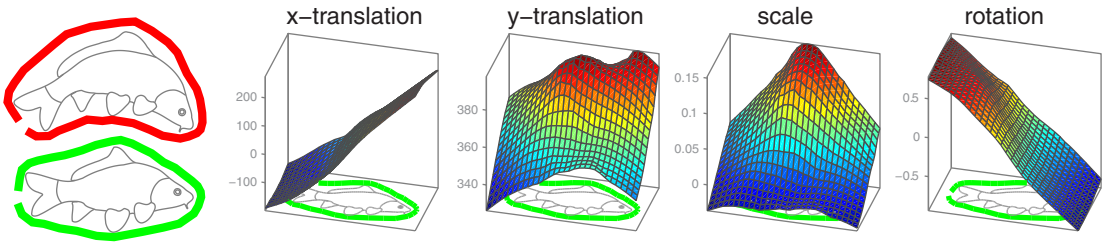


Figure 3.3: Regions with high transformation parameter similarity are related by a smoothly changing transformation function. The green region on the left is related to the red region by the transformation function shown on the right. Each of the four transformation parameters of the function is shown in a separate plot. Note that there are no discontinuities or spikes in the gradient magnitude of the parameter functions.

p is defined as $S(t_x, t_y, s, r) * p = S^L(t_x, t_y) * S^S(s) * S^R(r) * p$, where S^L, S^S and S^R are translation, scaling and rotation operations, respectively, and $*$ applies an operation to a point p.

A *first-order correspondence* (p, τ) matches two points in a vector image based on their local neighborhood, e.g. using a local descriptor. It is defined by the position of the first point p and the transformation parameters τ relating the local neighborhoods of the two points: $(p, \tau) = (p_x, p_y, t_x, t_y, s, r)$. The second point can be found as $\mathcal{S}(\tau) * p$. The set of all first-order correspondences of a shape is denoted $\boldsymbol{\mathcal{C}} \subset \mathbb{R}^2 \times \mathbb{T}$. Note that in the continous case, there are infinitely many first-order correspondences for any given shape. Due to the necessary discretization of the problem and associated numerical issues, only a finite noisy subset of $\boldsymbol{\mathcal{C}}$ can be found.

We model the transformation relating two shape regions $\mathbf{Q}, \mathbf{T} \subset \mathbb{R}^2$ with a *transformation function*:

$$f : \mathbf{Q} \to \mathbb{T} \tag{3.1}$$

The transformed region is $\mathbf{T} = \{S(f(p)) * p \mid p \in \mathbf{Q}\}$. A rigid transformation is described by a constant function, while all other functions describe non-rigid transformations. Each subset of correspondences $\mathcal{T} \subset \boldsymbol{\mathcal{C}}$ that contains one correspondence for each point in the region $\mathbf{Q}$ defines a transformation function that relates two shape regions with points having similar local neighborhoods. Figure 3.3 shows a transformation function relating the green region to the red region, i.e. transforming each point p in the green region with the transformation given by $f(p)$ yields the red region. The four dimensions of the function values are shown separately on the right. Since the red region is a smoothly bent version of the green region, the function does not exhibit discontinuities and the gradient magnitudes of its individual dimensions vary smoothly. Intuitively, transformation functions with low gradient magnitude in all parameters ensure that the transformed region is not distorted beyond recognition and retains local geometric detail, but still allow for strong non-rigid deformations over the whole region.

The Jacobian of the transformation function combines the gradients of the individual transformation parameters. The Frobenius norm of the Jacobian describes the magnitude of the combined gradients. More specifically, it is the norm of the vector of gradient magnitudes:

$$\|J_f\|_F = \left\|\left(\|\nabla f_{\tau_x}\|, \|\nabla f_{\tau_y}\|, \|\nabla f_{\tau_s}\|, \|\nabla f_{\tau_r}\|\right)\right\| \tag{3.2}$$

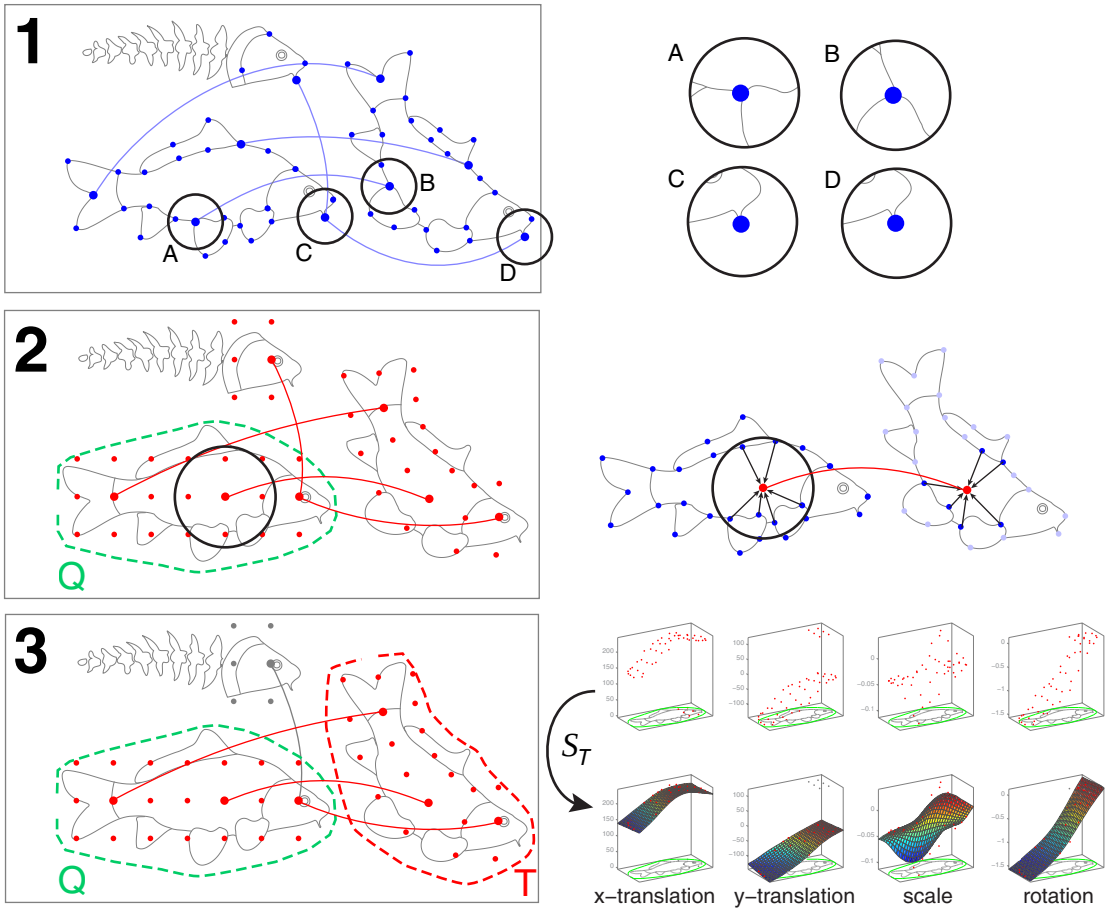


Figure 3.4: The three main steps of our method. From top to bottom: 1) First-order correspondences $\mathcal{C}$ (blue dots) are computed. To avoid clutter, only some correspondences are shown with connecting lines. 2) Refined first-order correspondences $\mathcal{C}'$ (red dots) are computed by aggregating nearby unrefined correspondences. 3) Transformation functions $f_{\mathbf{T}}$ are found in transformation space, corresponding target regions $\mathbf{T}$ that are similar to the query region $\mathbf{Q}$.

where J_f is the Jacobian of f, $\|\cdot\|_F$ is the Frobenius norm and ∇f_* are the gradients of the individual transformation parameters of S.

We define the transformation parameter similarity between two regions $\mathbf{Q}$ and $\mathbf{T}$ related by a transformation function f as the harmonic mean of the Jacobian magnitude $\|J_f\|_F$ over the entire function. The harmonic mean is used because a single large Jacobian magnitude in the transformation function is percieved as a strong discontinuity in the target region. Such a region is considered less similar than a region with large areas of average Jacobian magnitude.

Transformation functions that relate two shape regions are subsets of $\boldsymbol{\mathcal{C}}$. However, $\boldsymbol{\mathcal{C}}$ may contain multiple transformation functions, as well as correspondences that are not part of any transformation function. The goal of our method is to find subsets $\mathcal{T} \subset \boldsymbol{\mathcal{C}}$ that approximate functions with the properties described above. Each subset then describes a pair of regions with high transformation parameter similarity.

Algorithm 3.1: TPS Matching

Input : query region $\mathbf{Q}$,
contour segments $\{P_j\}$

Output : sets of first-order correspondences $\{\mathcal{T}_k\}_{k=1}^N$
describing regions with similarities $\{TPS_k\}_{k=1}^N$

1 compute first-order correspondences $\mathcal{C}$ on contour segments $\{P_j\}$ using local shape descriptors
2 refine first-order correspondences $\mathcal{C}$ to get $\mathcal{C}'$ using Algorithm 3.2
3 the sets of correspondences $\{\mathcal{T}_k\}$ and their similarities $\{TPS_k\}$ are found as smooth transformation functions in $\mathcal{C}'$ using Algorithm 3.3

3.3 Algorithm Overview

The input to our algorithm is a model of 2D shapes defined by contour segments. Such segments can be defined as polylines, obtained for example by vectorizing the output of an edge detector like Canny. Broken contours are not overly problematic here. However, the polyline geometry should not be too noisy (if so it can be gracefully smoothed) as the local curvature is used by our local shape descriptor. The second input is a user-provided 2D *query region*, defined by outlining or brushing on the canvas. The goal is to find all 2D shape regions that have a high similarity to the query region. Our method proceeds in three steps (cf. Figure 3.4 and Alogrithm 3.1).

In the first step, the first-order correspondences are computed using a local shape descriptor. We use a variant of Shape Contexts [7], but any descriptor invariant to similarity transforms can be used as long as the resulting correspondences are reasonably accurate. Since this first step is interchangeable and just a minor contribution, we refer to Appendix A for details.

In practice only a noisy subset of $\boldsymbol{\mathcal{C}}$ denoted $\mathcal{C}$ is found in the first step. We could find smooth transformation functions directly on this set of correspondences, but it is usually prohibitively large and may not cover all areas of the query region, degrading the stability of our method. To improve stability, we introduce an intermediate step that has three roles: it reduces the number of first-order correspondences to decrease computation time, generates correspondences that cover the query region more evenly and reduces noise. In this step, the query region is sampled regularly with a fixed number of points. For each of these points, we find stable correspondences by aggregating nearby noisy correspondences of $\mathcal{C}$ using the method of Mitra et al. [73], as described in Section 3.4. The result is a smaller set of first-order correspondences $\mathcal{C}'$ that contains less noise and covers the query region more evenly.

In the final step, we search for sets of correspondences $\mathcal{T} \subset \mathcal{C}'$ that are smooth transformation functions (cf. Section 3.2). We use linkage clustering with weights based on approximate Jacobian magnitudes to identify these functions, as will be explained in Section 3.5. Each function corresponds to a region similar to the query region.

Algorithm 3.2: Refine First-order Correspondences

Input : query region $\mathbf{Q}$,
noisy first-order correspondences $\mathcal{C}$
Output : refined first-order correspondences $\mathcal{C}'$

1 create query points $Q = \{q_k\}_{k=1}^{M}$ by sampling the query region $\mathbf{Q}$ in a regular grid
2 **for** $q_k \in Q$ **do**
3 compute weights $\{w_i\}_{i=1}^{|\mathcal{C}|}$ using Equation 3.3
4 compute set of transformation parameters $\{\tau\}_k$ as the dominant modes of the noisy correspondences $\mathcal{C}$ weighted by $\{w_i\}$ in transformation space $\mathbb{T}$
5 **end**
6 the refined correspondences $\mathcal{C}'$ are the pairs $\{(q, \tau)_j\}$ of all query points

3.4 Improving First-Order Correspondences

The first-order correspondences found by the local shape descriptors are usually noisy and may not cover the query region evenly. This makes finding smooth transformation functions more difficult. Consequently, we refine the original set $\mathcal{C}$ of first-order correspondences into a new set $\mathcal{C}'$ having fewer elements with less noise that are evenly spread over the query region.

To create the new set of correspondences, we first determine the locations q of the refined correspondences (q, τ). The locations q are constructed by placing a fixed amount of points ($\sim$50 in our examples) regularly in the bounding box of the query region and only the points inside the query region are kept. We call these points *query points*.

Next, the transformation parameters τ for each query point are found. The method of Mitra et al. [73] finds rigid region matches in a shape given a set of noisy first-order correspondences. We apply this method once for each query point q, using only the noisy correspondences near q as input, as we will explain in detail below. This effectively gives the rigid matches of the local query point neighborhood. The rigid transformation parameter τ of each match is used to construct a refined correspondence (q, τ).

Given a query point q, noisy correspondences (p_i, τ_i') in the set $\mathcal{C}$ are weighted by the following function around q:

$$w_i = G(||p_i - q||)\ h_i'\ \frac{1}{d_i} \tag{3.3}$$

where G is a Gaussian with a radius r_s defined by the user. The Gaussian can be thought of as a smoothing kernel. Increasing the radius gives smoother results, since each refined first-order correspondence is based on a larger set of noisy first-order correspondences. We use a radius of $0.1b$ in all our examples, where b is the length of the bounding box diagonal of the input shape. Each noisy correspondence might have a confidence value h_i'. If no confidence values are available, a factor of one is used instead. d_i is the density of the first-order correspondences in $\mathcal{C}$ computed at each point p_i. This factor removes the bias that would otherwise be introduced due to different densities of first-order correspondences over the shape. A straightforward way to compute the density at a point p_i is to center G at p_i and accumulate $G(||p_i - p_j||)$ for all points $p_j \in \mathcal{C}$ with $i \neq j$.

With the weighted noisy correspondences near the query point q as input, we now proceed according to the method of Mitra et al. [73]. Noisy correspondences with a weight smaller than one percent of the maximum weight are discarded. The transformation parameters τ_i' of the noisy correspondences weighted by w_i form a density distribution in transformation space $\mathbb{T}$. The dominant modes τ of this distribution correspond to transforms used by many noisy correspondences. Consequently, if such transforms are applied to the query point, it will be mapped to a neighborhood similar to the query region. Because no a-priori knowledge about the number of dominant modes is available, mean-shift clustering [21] with a kernel radius of one, due to the transformation space normalization, is used to find them.

The refined correspondences are the query points combined with each of their dominant modes:

$$\mathcal{C}' = \{(q, \tau)_j\} \tag{3.4}$$

where the index j runs over the dominant modes of all query points. Additionally, we use the magnitude h of the dominant modes as a confidence value for the refined correspondences. A higher magnitude means more noisy correspondences agree on the transformation parameters. The number of refined correspondences is the total number of dominant modes of all query points. The number of dominant modes for each query point depends on the complexity of the shape, as well as on the radius of the Gaussian in Equation 3.3. See Algorithm 3.2 for a summary of the refinement step.

3.5 Region Matching

We now proceed to the main step of our method: finding shape regions that have high transformation parameter similarity to the query region. A similar region is related to the query region by a transformation function with low Jacobian magnitude. Each refined first-order correspondence (q, τ) can be seen as a sample of a transformation function, where q is the function argument and τ the value. The set $\mathcal{C}'$ of refined first-order correspondences may contain samples for many transformation functions, as well as samples that are not part of any function. The goal in the final step is to identify subsets of first-order correspondences that approximate transformation functions with low Jacobian magnitude (cf. Appendix B for an example). Each subset contains correspondences between the query region and a region similar to the query region.

To find these subsets, we first define a second-order similarity for each pair of first-order correspondences. The second-order similarity between two first-order correspondences $(q, \tau)_i$ and $(q, \tau)_j$ is based on a lower bound for the Jacobian magnitude that any function passing through both correspondences must have. A lower bound for the gradient magnitudes of the individual transformation parameters $\tau = (t_x, t_y, s, r)$ is the vector of difference quotients:

$$\mathbf{a}_{ij} = \frac{(t_x, t_y, s, r)_i - (t_x, t_y, s, r)_j}{\|q_i - q_j\|} \tag{3.5}$$

Since the Jacobian magnitude is the norm of the vector of gradient magnitudes (cf. Equation 3.2), the norm of $\mathbf{a}_{ij}$ is a lower bound for the Jacobian magnitude. We define the second-order

Algorithm 3.3: Find Smooth Transform. Functions in $\mathcal{C}'$

Input : refined first-order correspondences $\mathcal{C}'$
Output : sets of first-order correspondences $\{\mathcal{T}_k\}_{k=1}^N$ describing regions with similarities $\{TPS_k\}_{k=1}^N$

1 initialize sets of correspondences $\{\mathcal{T}_k\}$ as $\{\mathcal{T}_k = \{(q,\tau)_k \in \mathcal{C}'\} \mid k = 1 \ldots |\mathcal{C}'|\}$
2 compute linkage weights between all pairs $(\mathcal{T}_i, \mathcal{T}_j)$ using Equation 3.10
3 **while** *highest linkage weight is above threshold* **do**
4 merge correspondence sets $\mathcal{T}_{i^*}$ and $\mathcal{T}_{j^*}$ having the highest linkage weight: $\{\mathcal{T}_k\} = (\{\mathcal{T}_k\} \cup \{\mathcal{T}_{i^*} \cup \mathcal{T}_{j^*}\}) \setminus \{\mathcal{T}_{i^*}, \mathcal{T}_{j^*}\}$
5 update linkage weights for the merged correspondence set
6 **end**
7 compute similarities $\{TPS_k\}$ for each set in $\{\mathcal{T}_k\}$ using Equation 3.11

similarity as the Gaussian of this lower bound:

$$a_{ij} = G(\|\mathbf{a}_{ij}\|) = G\left(\frac{\|\tau_i - \tau_j\|}{\|q_i - q_j\|}\right) \tag{3.6}$$

where a_{ij} is the second-order similarity between correspondence i and correspondence j. Note that the transformation parameters τ are normalized, as explained in Section 3.2, otherwise the difference quotient would depend on the scale of the individual parameters. The radius of the Gaussian is set to one. Changing the radius would have the same effect as changing the normalization factors.

Given the second-order similarity for all pairs of correspondences, we find subsets of correspondences that approximate functions with low Jacobian magnitude. Hierarchical average linkage clustering with linkage weights based on the second-order similarity is used to find these subsets. We start with each correspondence in a separate set and iteratively merge neighboring sets having elements with high linkage weights. The linkage weights between correspondences $(q,\tau)_i$ and $(q,\tau)_j$ are defined as:

$$l_{ij} = a_{ij}\, h_i h_j n_{ij} \tag{3.7}$$

where h are confidence values for correspondences i and j, effectively favoring sets of correspondences with high confidence. The factor n_{ij} is a neighborhood weight that restricts merging operations to neighboring sets of correspondences:

$$n_{ij} = G(\|q_i - q_j\|) \tag{3.8}$$

G is a Gaussian with a radius r_n that controls the trade-off between the proximity of sets and their second-order similarity. A low radius will only allow sets of correspondences to be merged that have adjacent query points. This precludes finding shape regions with occlusions or regions missing correspondences due to noise or bad matching. Setting the radius too high might result in correspondences from distant parts of the query region being merged, ignoring the correspondences in between, as shown in Figure 3.5. Unless noted otherwise, we use a value of $0.003b$ in all our scenes, where b is the length of the bounding box diagonal of the input shape.

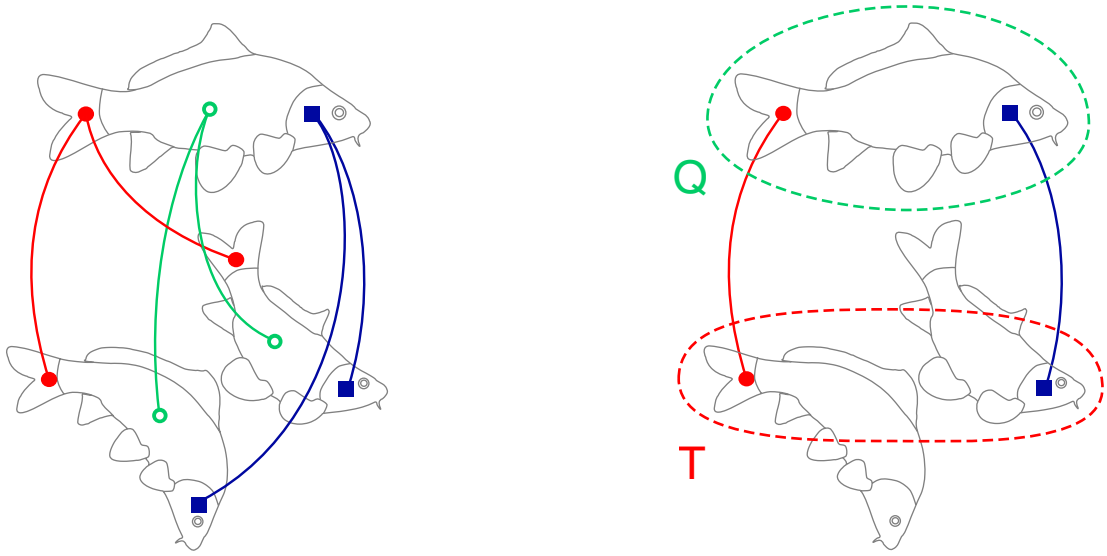


Figure 3.5: Effects of an overly large neighborhood radius. Three fish are related by the correspondences shown on the left. The tail of the lower fish has similar transformation parameters as the head of fish in the middle. An overly large neighborhood radius would allow our method to merge correspondences from distant scene parts, while the center correspondences (green) are ignored. This causes an incorrect matching, as shown on the right.

In each iteration, the two sets of correspondences $\mathcal{A}$ and $\mathcal{B} \subset \mathcal{C}'$ with the highest average of the linkage weights between their elements are merged. Each set contains samples of a transformation function that has low Jacobian magnitude when reconstructed from the samples. In some cases, such as branching structures that are matched to a non-branching query region, two sets of correspondences may get merged that have a large overlap in their query points. For example, two branches and the trunk of a tree may get matched to a query region that contains only a trunk and a single branch, if the trunk splits smoothly into the two branches. To pick only a single branch, we introduce a factor that encourages regions that contain exactly one correspondence of each query point q:

$$d_{\mathcal{AB}} = 2^{N^s_{\mathcal{AB}} - N^d_{\mathcal{AB}}} \tag{3.9}$$

where $N^d_{\mathcal{AB}}$ is the number of query points that are present in the correspondences of both sets and $N^s_{\mathcal{AB}}$ the number of query points present in only one of the sets. Two sets having a large overlap in the domain of the transformation function have a low fold-over factor. The final linkage weight between two sets of correspondences $\mathcal{A}$ and $\mathcal{B}$ is defined as:

$$l_{\mathcal{AB}} = \frac{\sum_{(i,j)_{\mathcal{AB}}} l_{ij} n_{ij}}{\sum_{(i,j)_{\mathcal{AB}}} n_{ij}} d_{\mathcal{AB}} \tag{3.10}$$

where $(i,j)_{\mathcal{AB}} = \{(i,j) \mid (q,\tau)_i \in \mathcal{A} \;\; (q,\tau)_j \in \mathcal{B}\}$. The algorithm stops merging the sets with highest linkage weight when this weight falls below a threshold. In our implementation we use one percent of the largest linkage weight.

Each resulting set $\mathcal{T} \subset \mathcal{C}'$ defines a transformation function $f_{\mathbf{T}}$ with low Jacobian magnitude that describes the transformation between the query region $\mathbf{Q}$ and a target region $\mathbf{T}$. Finally, the transformation parameter similarity of the two shape regions $\mathbf{Q}$ and $\mathbf{T}$ is approximated by the weighted harmonic mean (cf. Section 3.2) of the linkage weights between all pairs of correspondences in $\mathcal{T}$:

$$TPS_{\mathbf{QT}} = |\mathcal{T}| \left(\frac{\sum_{(i,j)_{\mathcal{T}}} l_{ij}^{-1} n_{ij}}{\sum_{(i,j)_{\mathcal{T}}} n_{ij}} \right)^{-1} \tag{3.11}$$

where $(i,j)_{\mathcal{T}}$ are the indices of the pairs of correspondences. See Algorithm 3.3 for a summary of the region matching step.

3.6 Results and Discussion

We implemented the described method in Matlab and tested it on a PC with 16 GB memory and a 3.4 GHz quad-core CPU. Key parts of the correspondence refinement step and the clustering step were implemented in CUDA or C++. The first step (cf. Section 3.3) creates first-order correspondences using local shape descriptors (cf. Appendix A). This step is computed once for each shape in a preprocess and is independent of the query region. Steps two and three are computed after a user selects a query region and depend mainly on the number of query points (usually $30 - 50$ in our examples). The linkage clustering described in Section 3.5 has a worst-case complexity of $O(N^3)$ in the number of refined first-order correspondences, but the average case has much lower complexity. Additionally, we discard pairs of refined first-order correspondences with very low confidence (below one percent of the maximum confidence) before clustering. The computationally most intensive case for our method is a shape where all pairs of points have the same similarity, e.g., a circle. Since the stopping criterion of the clustering step would never be met, the clustering algorithm would need run through N iterations to merge all correspondences, searching for the best among $(N-k)^2$ correspondence pairs in step k.

The method is demonstrated on several vector images shown in Figures 3.6 and 3.7. The images are created by automatic contour finding from RGB-images with subsequent manual noise removal (h), manual drawing (c and i) or a combination of both (a,b,d,e,f,g). In each example, we perform 1-3 operations. An operation includes finding all red target regions matching the green query region with a TPS above a given threshold (which is selected interactively with a slider after the region matching step has finished) and then transferring the texture shown in the query region to each target region to visualize matching accuracy. Each texture has an alpha-channel and is added on top of the existing textures. The background is only included to give a better impression of how the final result of an artist-created image using our method might look like. The only input to our region matching method are the shape contours in the leftmost column.

In Figure 3.1, the small snake is related to the big snake by a transformation function with changing scale, angle and translation parameters. The skin and the pattern of the snake are transferred in two operations. Figures 3.6a and 3.7g show similar images with a significant amount of clutter. Nevertheless, it is possible to detect small objects like the leaf in Figure 3.7h or the small clovers in Figure 3.6a. In Figure 3.6b, all fish in the image have different proportions and additionally, some are bent. The contours of some fish have large gaps due to occluding

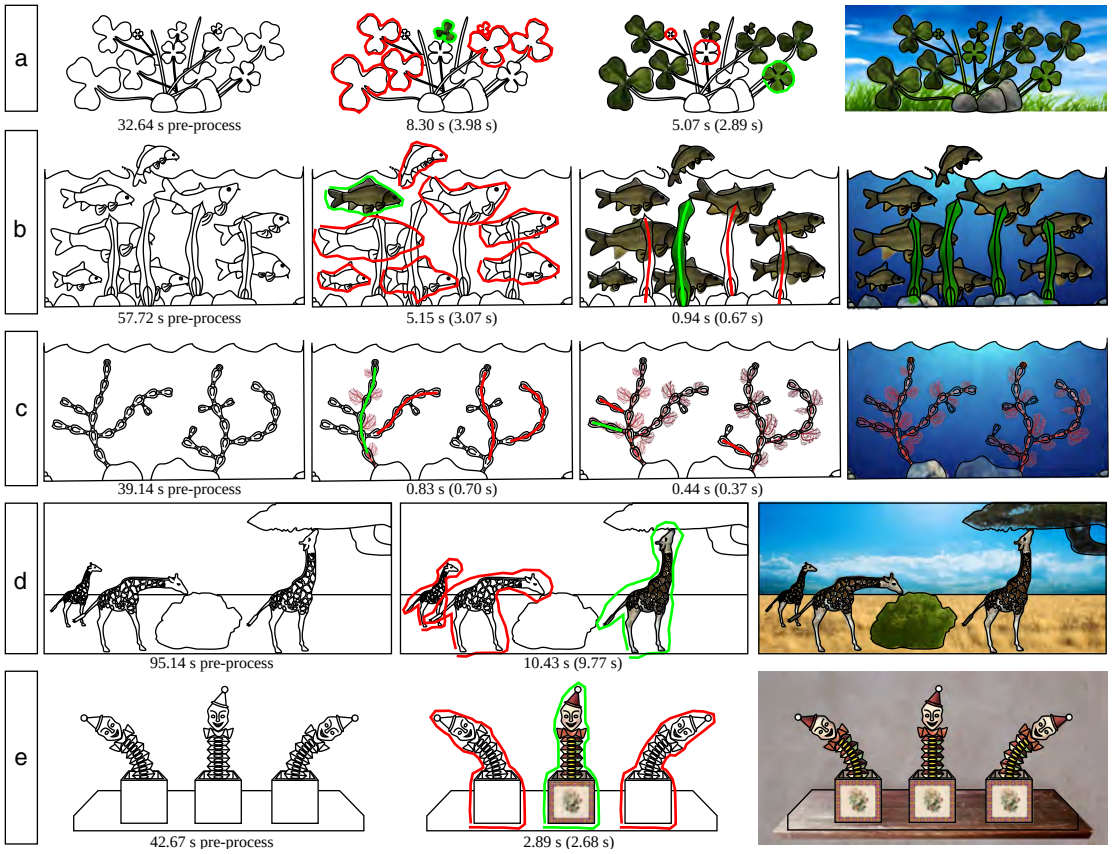


Figure 3.6: Results of our method on five test shapes. The leftmost image shows the input shape. The next image shows one operation where region matches (red) are found for a user-defined query region (green). The texture from the query region is transferred to each such target region, as shown in the following image. The final result is shown in the last image with added background to give an impression of how an artist-created result might look like when using our method. Timings are provided below each image (see Section 3.6 for details).

sea weed. For the sea weed we use query points on a stroke instead of a region. The texture is transferred by extrapolating the texture coordinates of the matched (red) strokes. Figure 3.6c shows branching corals. Due to the thin structure of the corals, strokes are used here instead of regions. In 3.6d, giraffes in different poses are matched by our method. Figure 3.6e shows man-made objects with different amounts of bending (also note that the hat of the left harlekin is mirrored), while in 3.7f two snakes with a fine skin pattern are matched. In Figure 3.7g, eight different types of butterflies are matched in an image with a large amount of background clutter. The scale difference between butterflies has a factor of up to ten. Finally, Figure 3.7i shows a stylized sun with non-rigidly transformed rays being constructed in three operations (excluding the solid-colored background).

Detailed timings for each operation are shown in Figures 3.6 and 3.7 below the image of the corresponding operation. The time in parenthesis is for the refinement step, the time outside

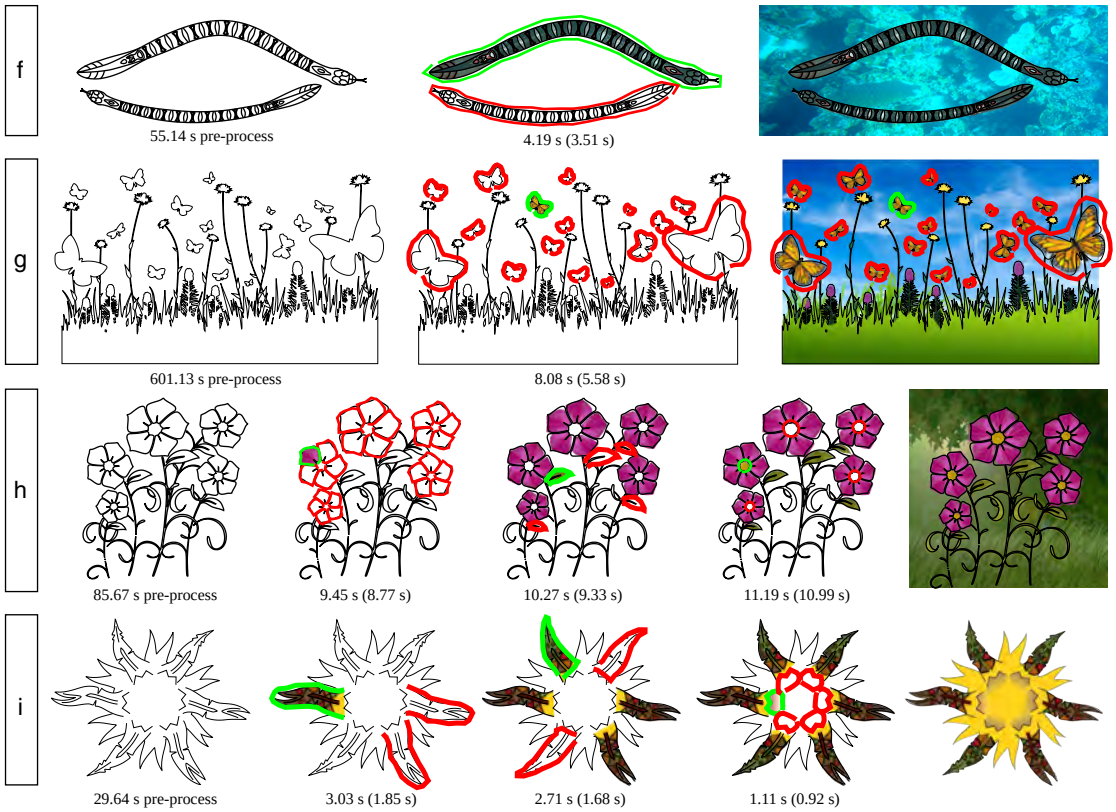


Figure 3.7: Additional results. The leftmost image shows the input shape. The next image shows one operation where region matches (red) are found for a user-defined query region (green). The texture from the query region is transferred to each such target region, as shown in the following image. The final result is shown in the last image with added background to give an impression of how an artist-created result might look like when using our method. Timings are provided below each image (see Section 3.6 for details).

the parenthesis for the entire operation. Times for the first step of our method, computed as a pre-process, are shown below the input image. The largest factor for the computational demand is the number of refined correspondences, which mainly depend on scene clutter. For this reason, the operations in the flowers scene are computationally most expensive, followed by the butterflies and the clovers scene. In the giraffes example we used more query points than in the other examples to better capture the thin legs, making the operation more expensive.

Although the refinement step improves first-order correspondences considerably, the inlier percentage is still relatively small. Table 3.1 shows the inlier percentage for the first operation in all example images. In Figure 3.8 we show the distribution of refined first-order correspondences in the snakes image (Figure 3.1) in transformation space. The region matching step correctly finds the matching snake in the presence of a significant amount of outliers. Note how the matched region consists of high-confidence as well as low-confidence correspondences, precluding simple thresholding. See Appendix B for additional and more detailed visualizations.

result	corr. count	inlier %	weighted inlier %
3.1	2551	2.47	4.18
3.6a	6648	5.91	17.95
3.6b	7810	4.34	19.65
3.6c	278	7.91	7.39
3.6d	5664	2.31	23.98
3.6e	2028	5.42	16.60
3.7f	2999	1.87	4.62
3.7g	5668	12.35	29.69
3.7h	1911	55.78	88.17
3.7i	4005	2.42	14.02

Table 3.1: Inlier percentage for the first operation in all examples shown in Figures 3.1, 3.6 and 3.7. Columns from left to right: total number of refined first-order correspondences, percentage of inliers thereof, percentage of inliers weighted by their confidence (see Section 3.4).

We compare our method to the L_2-Estimator (L2E) [66] Agglomerative Correspondence Clustering (ACC) [19] and the Non-Rigid Dense Correspondence (NRDC) method [43], three state-of-the-art methods for non-rigid region matching. For all three methods, we optimize the parameters for each image separately to provide the best matching. The correspondences found by these methods may not cover the entire query region, making it necessary to extrapolate the missing parts to establish a dense correspondence. For each image, we choose either Locally Linear Embedding [86], thin-plate splines or a Gaussian-weighted average of the transformation parameters for the extrapolation, whichever gives the best results. In our method, the correspondence refinement step makes a separate extrapolation unnecessary. To validate the results, we use a ground truth generated by manually placing a sparse set of correspondences at key locations of the shape (cf. Figure 3.9d). Since the L2E method works on point sets, we use evenly-spaced samples on the contour segments as input. The ACC and NRDC methods both require images as input. We transfer the texture in the query region to all target regions using the ground truth and use the resulting image as input. The area inside the query region is used as source image and the area outside the query region as target image. We also tried using rasterized contours as input, but using textures gave better results for all scenes. The input for each method is shown in the insets of Figure 3.9.

The L2E method alternates between improving the transformation and the correspondence between the source and target point sets. The transformation is modeled as a function restricted to lie in a specific reproducing kernel Hilbert space and is estimated using their robust L_2 estimator. Correspondences are updated in each iteration using the Hungarian method to connect points with the most similar shape contexts. This method is designed for two point sets that have a single best match and exhibits two main problems in the presence of multiple best matches. First and most importantly, the method might get stuck in a local minimum if there are multiple basins of attraction, since one subset of the points might be initialized in one basin and a second subset in a different basin. The result of this behavior can be observed in rows 1, 5 and 6 of Figure 3.9a. Note

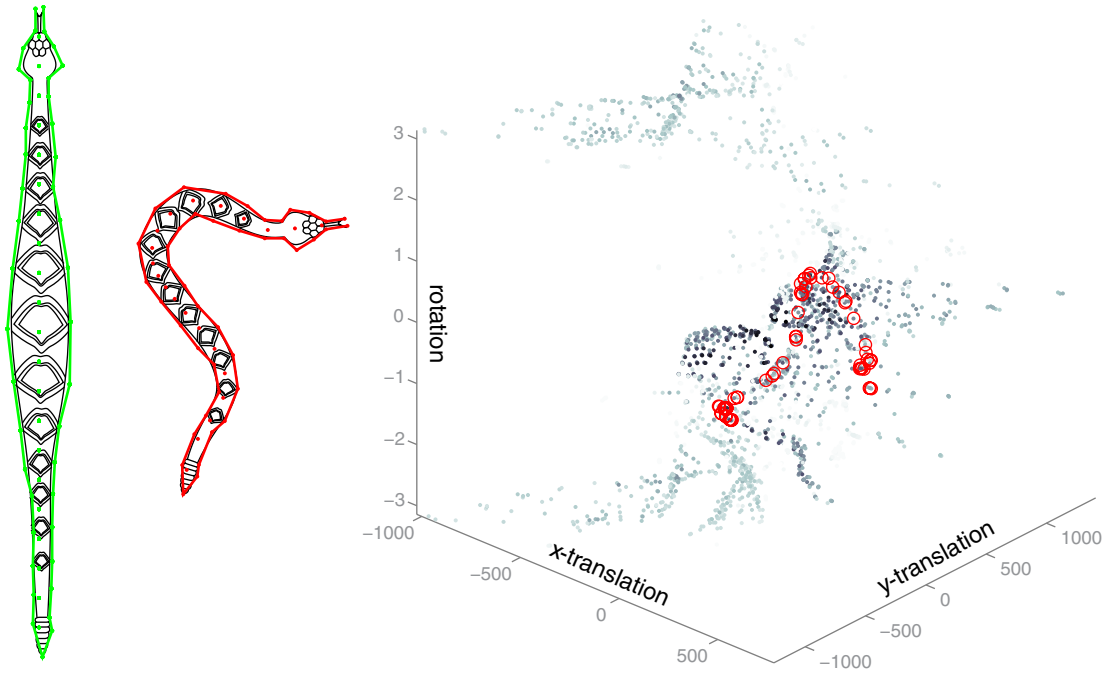
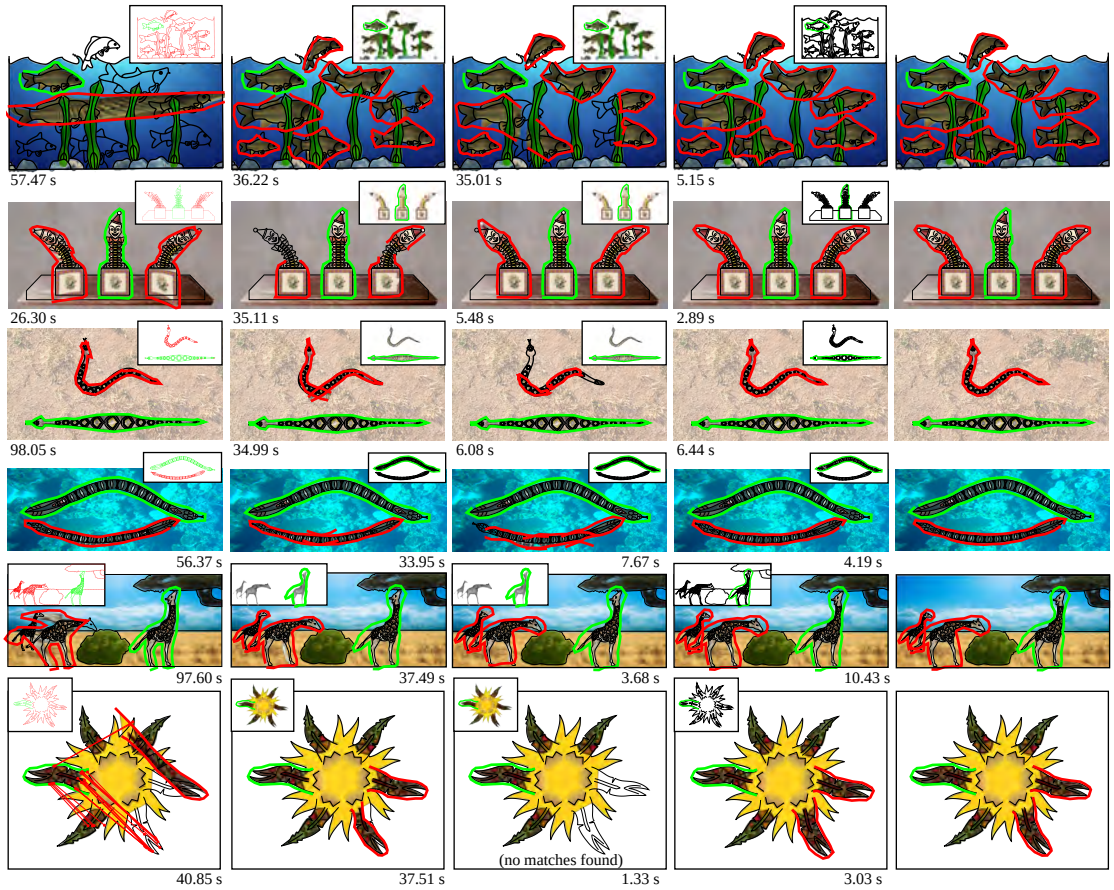


Figure 3.8: Refined first-order correspondences of the snakes image (Figure 3.1) in transformation space. Only the rotation and translation dimensions are shown. Dark correspondences have high confidence, light correspondences have low confidence. Inliers corresponding to the red target region are encircled in red.

that we only show one matched region in the first and last row to avoid clutter, since the other regions are distributed over a larger number of attractors resulting in an even larger distortion. In the last row we additionally show the correspondences found for the second matched region as red lines. The second problem is that making the shape contexts scale and translation invariant is not trivial anymore if the matching regions are not centered in their point set. The authors use the average distance of the point set to the shape context center as scale and the direction of the shape context center to the point set center of mass as direction of a shape context. These approximations are only valid if both the source and target regions have approximately the same scale and are centered in their point sets. Using the contour normal and curvature information, as we do, is not possible, since the source points are transformed in each iteration and the contour information is invalidated. Badly rotated shape contexts result in the mismatched head of the harlekin in row 2 of Figure 3.9a.

The ACC method clusters first-order correspondences based on the absolute difference between their transformations. This means that sparse correspondences on a strongly non-rigid region, like on the 'neck' of the harlekin or the body of the snake in Figure 3.9b, are considered dissimilar and the matched region breaks apart. Additionally, the correspondences found by the method are sparse and only cover part of the query region. In contrast, our method uses the *gradient* of transformation parameters instead of the absolute difference, making it possible to correctly identify regions defined by a sparse set of first-order correspondences (cf. Figure 3.9c).



(a) L2E (b) ACC (c) NRDC (d) Our method (e) Ground Truth

Figure 3.9: Comparison to the L_2-Estimator (L2E) [66], Agglomerative Correspondence Clustering (ACC) [19] and the Non-Rigid Dense Correspondence (NRDC) [43] methods. The red target regions are matches found for the green source region. A texture is transferred from source to target regions to visualize the matching accuracy. The input to the individual methods is shown as inset. A ground truth created by manually placing key correspondences is provided as reference. Note that L2E has problems with multiple matching regions, ACC has problems with regions related by strongly non-rigid transformations and NRDC has problem with occlusions, repeating patterns and regions without detailed texture. Our method can handle multiple matches, occlusions, strong non-rigid transformations and repeating patterns making it possible to correctly detect all similar regions.

The NRDC method constructs first-order correspondences by finding a single best match for each $12x12$ pixel patch in an image using Generalized Patch Match [5] and then grows regions by merging adjacent correspondences based on the absolute difference of their transformations. Since the correspondences densely cover the query region, using the absolute difference does not create the same problems as in the ACC method. However, only merging adjacent correspondences precludes the handling of occlusions, as shown in Figure 3.9c, first row. Also, the single best match for each image patch is found based on local information only. Each element of a repeating

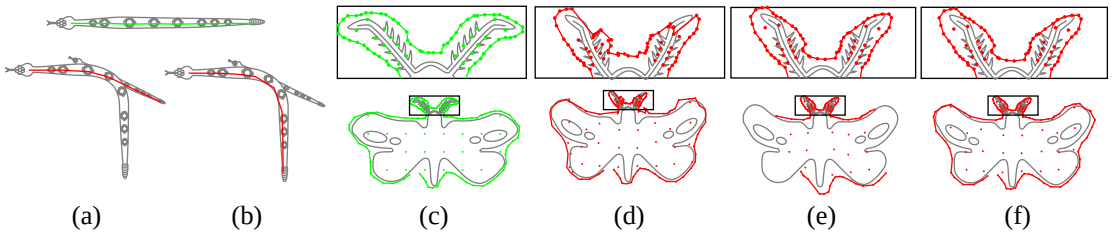


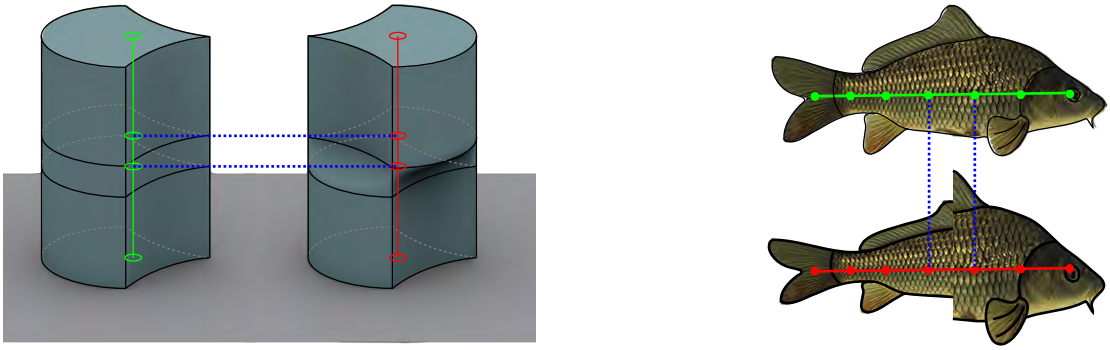
Figure 3.10: Two problems when using a second-order similarity based on the projection error. In (a) and (b), a snake is matched to two snakes having different scales. The similarity based on the projection error (a) fails to separate the two snakes, whereas our similarity measure (b) can correctly separate the snakes based on the scaling parameter of their transformations. The butterfly in (c) is matched non-rigidly to the butterfly shown in (d)-(f). The similarity based on the projection error either incorrectly merges correspondences on the feelers (d) or fails to match the body (e), depending on parmeter settings. Our method (f) can correctly match body and feelers using standard parameters.

pattern, such as the pattern on the snake skin (cf. Figure 3.9c, third and fourth row) is locally similar to all other elements, causing mismatches that can only be resolved when using global information, i.e. the position of the pattern element on the snake. Our method finds several matches for each query point and chooses a subset of these matches based on global information, correctly matching regions with repeating patterns. Finally, since NRDC densely computes correspondences for each pixel based on a relatively small neighborhood, it only works on regions having detailed texture. Less detailed textures (see bottom row of Figure 3.9c) can not be handled.

Relation to the Projection Error The projection error is a standard method of measuring second-order similarity. Our method can be adapted to use the projection error by changing the Jacobian magnitude in Equation 3.5 to the projection error as defined in Equation 1 of Cho et al. [19] or Equation 1 of HaCohen et al. [43]. However, we argue that the TPS has two main advantages over the projection error when computing the similarity between two first-order correspondences:

First, the projection error is computed in the 2D space of positions, whereas transformations usually have a much larger number of free parameters (four in our case), and information is necessarily lost in the projection of the transformation parameters into the 2D space. Consequently, the distance in the 2D space does not always correlate with the distance of the transformation parameters. Figure 3.10a shows two snakes with clearly different scales, but the second-order similarity based on projection error does not have enough information to distinguish between the two scales and incorrectly merges them. This problem gets worse as the number of transformation parameters increases. For example, when using non-uniform scaling or when extending the method to 3D input, completely different transformations might have zero projection error. We illustrate this problem in Figure 3.11. In contrast, since our method directly compares the transformation parameters, no information is lost and the scale difference in Figure 3.10b is correctly detected.

Second, TPS is gradient-based, whereas the projection error uses an absolute distance, making it less suitable to handle distributions of query points with varying densities. The projection



(a) (b)

Figure 3.11: Projection error used with transformations having many free parameters (note that this figure is only meant to illustrate a potential problem, we did not implement 3D transformations or non-uniform scaling). In both examples, the projection error between the two correspondences shown as blue dotted lines would be zero, even though their transformation is clearly different. In (a) the green and red strokes at the center axis of the two extruded shapes would be matched even though the second shape has a large discontinuity in its rotation around the center axis and in (b) the strokes inside the fish would be matched, even though the bottom fish has a large discontinuity in its vertical scaling.

error gets smaller as the density of correspondences in a region increases, even though the transformation of the region remains the same. Figure 3.10c-f shows two butterflies that are related by a non-rigid transformation. To accurately match the small feelers, they have to be sampled more densely. Consequently the projection error in the feelers is smaller than in the remainder of the body, causing either incorrectly merged correspondences in the feelers due to a small projection error (Figure 3.10d), or failure to merge correspondences in the body due to a large projection error (Figure 3.10e). This depends on the value of the normalization factor described in Section 3.2. The TPS is independent of sampling density, as it uses the gradient over the query region and can find a good match for both feelers and body using the standard value of the normalization factor (see Figure 3.10f).

Possible Extension to 3D Geometry One interesting direction for future work is the extension of our method to 3D geometry. In this case, to find first-order correspondences, we would need 3D surface descriptors such as Spin Images [46], Shape-DNA [84] or Heat Kernel Signatures [92] instead of our 2D descriptors. Using 3D descriptors might reduce false correspondences, since the local neighborhood of a point on a 3D surface is typically more discriminative than the local neighborhood on a 2D surface. Additionally, the transformation space $\mathbb{T}$ would be higher-dimensional, but this would pose no problem, as all employed techniques are well suited for high-dimensional data and have a linear dependence on the number of transformation parameters, both in execution time and memory consumption. A larger number of dimension in $\mathbb{T}$ might even be beneficial, since the correspondences in $\mathbb{T}$ are sparser, effectively reducing clutter in the transformation space. Intuitively, the transformation parameters should be smooth in the 3D case as well, however this would need to be shown empirically on a number of examples.

An interesting design choice would be the placement of query points. Query points could be

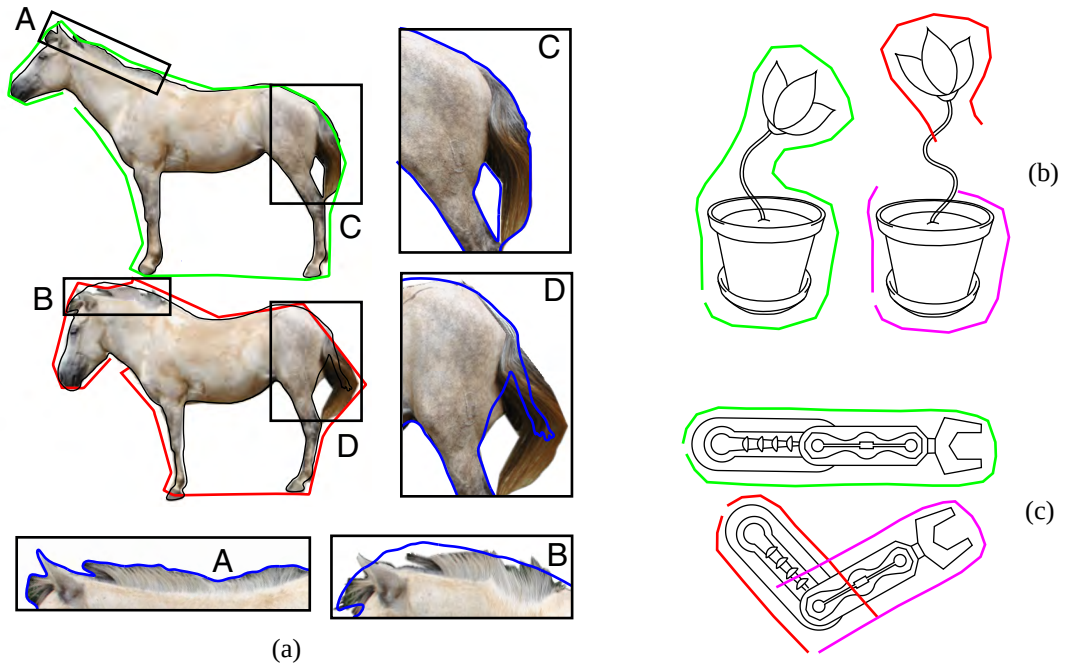


Figure 3.12: Limitations of our method. Example (a) shows the result of matching two shape regions with partially dissimilar geometry, as shown in the insets. The matches in the dissimilar parts are not accurate, but similar parts of the shape are still matched correctly. In example (b), the stems of the flowers do not have any distinctive features in common. Consequently, no first-order correspondences are found for the stems and our method can only match the flower blossom and pot. Example (c) shows two regions that are related by a discontinuous transformation function. Our method only matches its continuous subsets.

placed either in the volume or on the surface of a 3D shape. The first option would be analogous to our current method and would treat 3D regions as solid objects with a volume that cannot have strong discontinuities in its transformation. The second option would treat regions as 2D hulls and not constrain their volumes in any way, allowing for different kinds of transformations, such as a smooth and detail-preserving unfolding of open meshes. When using the second option, we would need to change the neighborhood weight in Equation 3.8 to approximate the geodesic distance on the shape surface. The differences to isometric shape matching discussed in Section 2.1 would also hold for a 3D extension of our method, such as the ability of our method to handle smooth changes in scale or more generally transformations that do not preserve pairwise distances between surface points.

The main challenge in implementing the volumetric version of our method might be performance, since the number of query points in the volume might get quite large. Another challenge is finding good local descriptors that provide reliable transformation parameters.

Limitations In the remainder of this section, we will discuss limitations of our method. First, due to our definition of similarity, our method can only match shape regions related by a smooth transformation function. Regions like joints have large gradients in the transformation parameters and can therefore not be matched, as illustrated in Figure 3.12c. Consequently, our method is

better suited for organic shapes or shapes that deform smoothly. Note that some of these smooth deformations like a gradual change in scale (cf. Figure 3.1) cannot be handled by other popular similarity measures such as isometry (as discussed in Section 2.1).

Second, similar regions need to have distinctive local features that can be matched by local descriptors. Otherwise first-order correspondences cannot be found. A failure case is shown in Figure 3.12b. The stems of the two flowers cannot be matched correctly, because locally, all parts of the two stems have equally similar geometry. Another reason for missing correspondences is dissimilar geometry. Figure 3.12a shows two horses with geometry that is mostly similar, except for the neck and tail as shown in the insets. These parts are missing first-order correspondences. In our implementation we try to infer missing correspondences from parts that have been matched correctly, but shape geometry in the missing parts cannot be taken into account.

3.7 Summary

In this chapter we have demonstrated that partial non-rigid shape matches can be found in 2D shapes by searching for sets of correspondences with smoothly changing scaling, rotation and translation parameters. The similarity measure resulting from this notion is a novel geometric analogy, which we call 'Transformation Parameter Similarity'.

Evaluations and comparisons to previous methods show that this measure can be used to match highly deformed shapes that are difficult to match with previous methods given only sparse and unevenly sampled point correspondences. Our method can match non-rigid shapes that are similar under our similarity measure and establish a dense mapping between pairs of matching shapes, even if the regions are only given as cluttered line segments. This mapping allows transferring texture or brush operations between two regions. Additional edit propagations, like transferring deformation operations to similar shapes, are possible using our method, and we would like to explore these applications in future work.

CHAPTER 4

Single-Example Geometric Analogies for Shape Arrangements

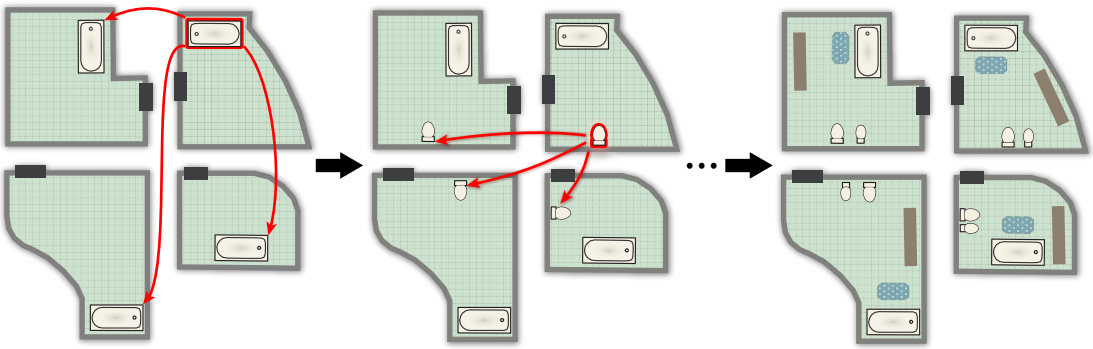


Figure 4.1: In this chapter, we present a method to propagate edit operations in 2D scenes. Objects can be propagated between general polygons of arbitrary shape while still giving plausible results. Our method can be used to place a large number of objects in 2D layouts using relatively few edit operations.

4.1 Introduction

In the previous chapter, we presented a method to transfer edit operations between deformable shapes, based on a custom similarity measure. Edits are applied to parts of the shapes that are locally similar. We now turn to the problem of edit propagation in scenes consisting of shape arrangements, such as floor plans. In contrast to the previous chapter, the arrangements need not be related by any kind of smooth transformation. The question of edit propagation can be posed for these scenes as well: when a user applies a change to a given scene, how can other, similar changes be applied to the scene automatically? This touches on the question of *shape similarity*

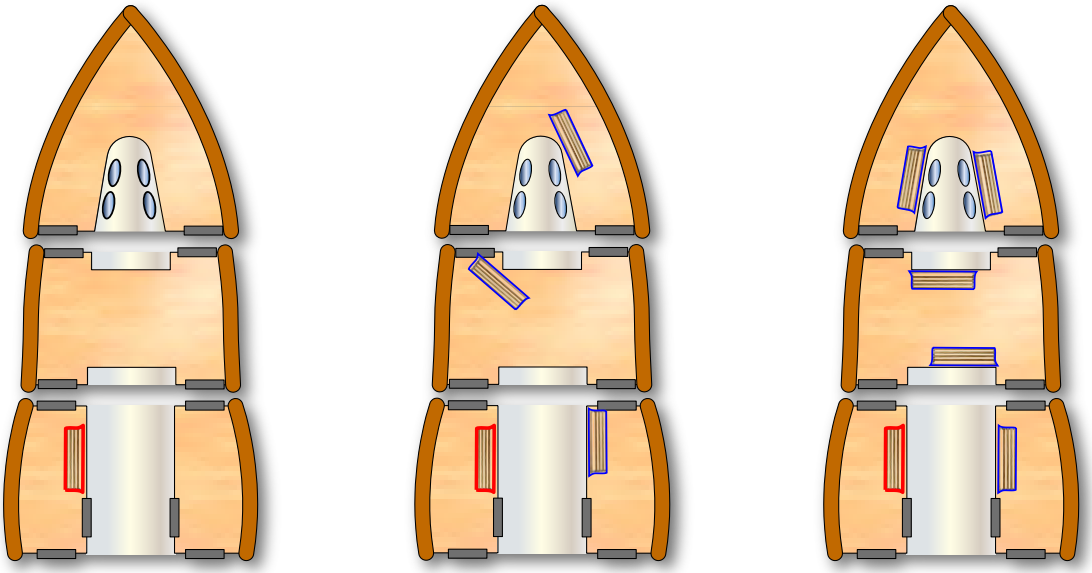


Figure 4.2: To find good positions for a bench (red) on the deck of a ship, many geometric relationships to the deck and the objects on the deck have to be taken into account. A naive approach using a local coordinate system based on boundary distance and boundary arc length (center – for more information, see Section 4.7) does not result in acceptable positions. Our approach (right) is flexible in its use of relationships, giving better positions.

and *coordinate systems*: One approach to effect similar changes is to apply the same operation to parts of the scene that are *similar* to the part that has been edited; furthermore, we need to find the exact *location* where the change should be applied in a particular similar part. The first problem, finding similar shape parts, is not always applicable to the types of scenes we are focusing on in this chapter. For example, the rooms in a floor plan may not always be similar, but similar operations need to be applied to them nonetheless.

For the second problem, different "local" coordinate systems have been proposed in animation and geometric modeling. These coordinate systems can be used to transfer a position in one polygon to another polygon of different shape. Examples include mean-value coordinates [44] and harmonic coordinates [47]. However, these systems work best in shape interpolation, where the basic topology of the shapes that provide the reference frame varies only by a little. Also, coordinate systems typically encode the notion of *distance*, but are not flexible enough to represent other geometric relationships.

In this chapter, we present a new method for edit propagation in 2D scenes that is more flexible and general than previous methods and can be used to propagate edit operations between dissimilar arrangements of shapes. Our method is based on the notion of *geometric relationships* between shapes and so-called *poses*, which we define as a location with an associated direction vector. While, for example, 2D Cartesian coordinates uniquely associate two values (coordinates) with every point in the plane, which represent the distance from one of the coordinate axes, our method can handle an arbitrary number of geometric relationships, and the mapping to points in

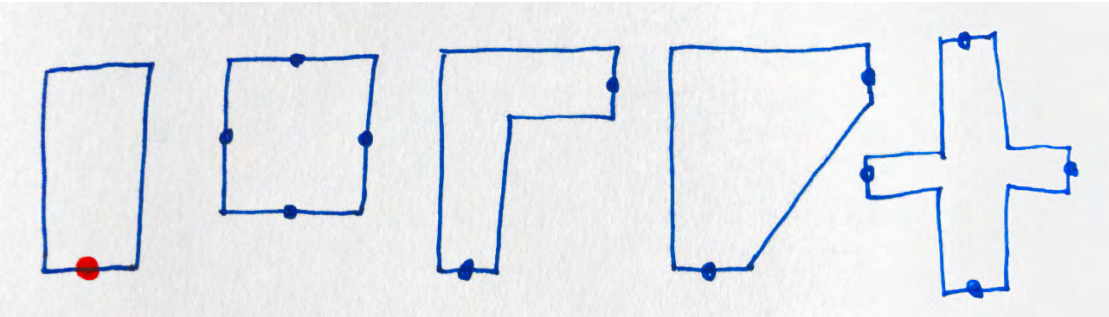


Figure 4.3: A human intuitively finds the blue point correspondences in the output polygons for the red point specified in the input polygon.

the plane need not be unique. This allows geometric relationships that do not depend on distance alone (like the relative direction with respect to a feature in the scene) to be introduced, and abandons those concepts from coordinate systems that are not required in our setting of edit propagation (see Figure 4.2).

Our current setup focuses on object placement, which is why we choose poses as the main interaction element. A pose can represent an object (e.g., a furniture item) and its orientation in a room. Furthermore, we concentrate on geometric relationships as structuring elements, rather than on similarity between the underlying polygons. The similarity of geometric relationships is effectively a type of geometric analogy, and using this analogy often gives plausible matches even if the degree of similarity is low, as different types of geometric relationships can be taken into account. Figure 4.1 presents an example of the proposed work flow.

4.2 Overview

The starting point for this work was our observation that most humans have an intuitive idea of how to transfer points between polygons. In Figure 4.3, for example, a human would be likely to pick the blue points as correspondences to the specified red point. Currently there is no computational tool available that could do the same. Generalizing this research problem to include directions and multiple polygons, we obtain the following research question:

Given a scene consisting of a set of polygons $\mathbf{A} = A_1, \ldots, A_n$, and a query pose $Q = (Q_p, Q_d)$, defined by a point and a direction, determine a number of poses $P_1, \ldots, P_n$ that have a local neighborhood that is "similar" to Q.

In Section 4.3, we introduce our notion of similarity. It is based on so-called *relationship functions*, which assign a numeric value to a certain *geometric relationship* between a pose and a feature in the scene. Features can be the polygons themselves, their individual corners, a certain segment, etc. Examples of relationship functions include the distance of query point Q_p to a polygon, or the angle by which Q_p is offset from the bisector of a polygon corner.

Given a query pose and a set of polygons, our algorithm has the following steps (see Figure 4.4):

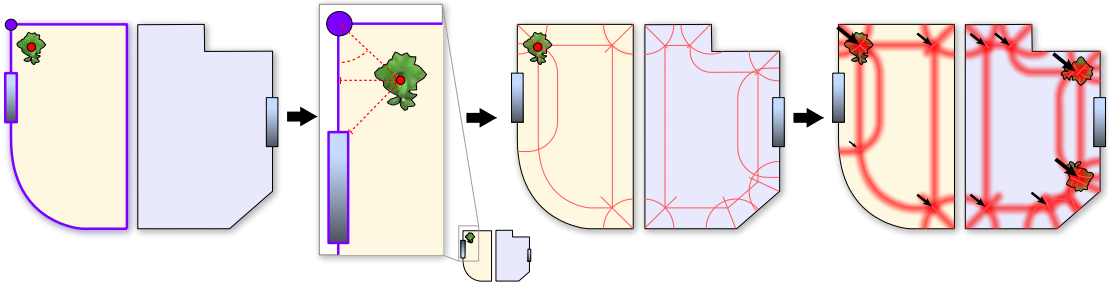


Figure 4.4: The main steps of our method. First, a query point is placed (plant) and important features are identified (purple). Then the relationship functions to the important features are evaluated (red dotted lines). Level sets for each feature and relationship function are constructed (red lines) and accumulated in pose space. The resulting maxima correspond to poses that have similar relationship function values as the query point.

1. Identify features in the scene (Section 4.3).
2. Evaluate the relationship functions (Section 4.3) for selected features, called *input features*.
3. For each relationship function and each feature, find the locations/poses in the scene that would give a similar function value as the query point. Basically, this corresponds to constructing a level set in "pose space" (Section 4.5) for each relationship function around a feature that is "compatible" to the input feature – we call these the *output features*.
4. Combine the level sets in pose space based on two weighting factors (Section 4.4):
   - the *local importance*, i.e., how relevant is the input feature to the query point for a given relationship function, and
   - the *matching accuracy*, i.e., how closely does the output feature match the input feature.
5. Find local maxima in pose space (Section 4.5) – these correspond to the desired poses that share as many of the defined relationships with the query pose as possible.

We also describe an extension to handle mirror symmetries present in the scene geometry (Section 4.6), and we describe how specific geometric relationships can be selected to form local coordinate systems with respect to features in the scene (Section 4.7). The application case of floor-plan editing is described in Section 4.8, which also includes useful extensions like probabilistic placement and feature selection, labels and hierarchies to restrict matching, and pose validation to avoid intersections in the result.

4.3 Geometric Relationships

In this section, we formally introduce the concept of geometric relationships through the definition of geometric features and relationship functions. As discussed in Section 4.2, we start with a

scene $\mathbf{A}$ and a query pose Q in a pose space $\mathbf{P}$. The pose space is the set of poses of interest, usually encompassing the scene and all angles from 0 to 360 degrees.

Features

Geometric relationships operate on features; the first step is therefore to extract features from the scene. The main features are the polygons in the scene themselves and the elements of the boundary of the polygon. We assume that the polygon boundary consists of "smooth" segments, i.e., sequences of edges with non-sharp angles, connected by "sharp" corners. We therefore use the following features:

- a **polygon** A, defined by a sequence of vertices $v_k, k \in \{1, \ldots, n_A\}$,
- a polygon **segment** $s_j \in A$, defined as a sub-sequence of consecutive smooth vertices $s_j = v_a, \ldots, v_b, a, b \in \{1, \ldots, n_A\}$. A vertex is called smooth if the dot product of its adjacent edges is above a threshold ($\frac{v_{k+1}-v_k}{||v_{k+1}-v_k||} \cdot \frac{v_k-v_{k-1}}{||v_k-v_{k-1}||} > \epsilon$), otherwise it is called sharp.
- a **corner** c_j of a polygon, defined by a vertex v_{k_j} that is shared by two adjacent segments s_{j-1} and s_j. Corners correspond to sharp vertices. In practice, the *bisector* c^b of the corner and the opening angle c^α will be relevant, as they are related to the direction vector of the query pose.

Relationship Functions

Given a set of features $\mathbf{F}$ and pose space $\mathbf{P}$, we define a relationship function as a functional on features and poses:

$$R : \mathbf{P} \times \mathbf{F} \rightarrow \mathbb{R}.$$

Each relationship function expresses a geometric relationship between a feature and a pose in a numeric way. Relationships include the distance of a pose from a feature, but also the angle between a pose and the feature. In the following, we list the geometric relationship functions that we use (see Figure 4.5). For convenience, we define the angle between vectors as $\angle(a, b) = \arccos(a/||a|| \cdot b/||b||)$.

- The **boundary distance** is defined as the minimum distance between the query point and a polygon boundary: $R_{bd}(Q, A) = \min_{x \in b(A)} d(Q_p, x)$, where $b(A)$ is the boundary of A.
- The **segment arc length** is the normalized arc length between the location of minimum distance to the query point and the start of the segment. If $x = \arg\min_{x \in b(s_j)} d(Q_p, x)$, then $R_{sl} = \frac{t_x - t_a}{t_b - t_a}$, where t_x is the arc length at x and t_a, t_b the arc length at the start and the end of the segment, respectively.
- The **segment distance** $R_{sd}(Q, e)$ is the same as the boundary distance, but considers only a segment $s \in A$.

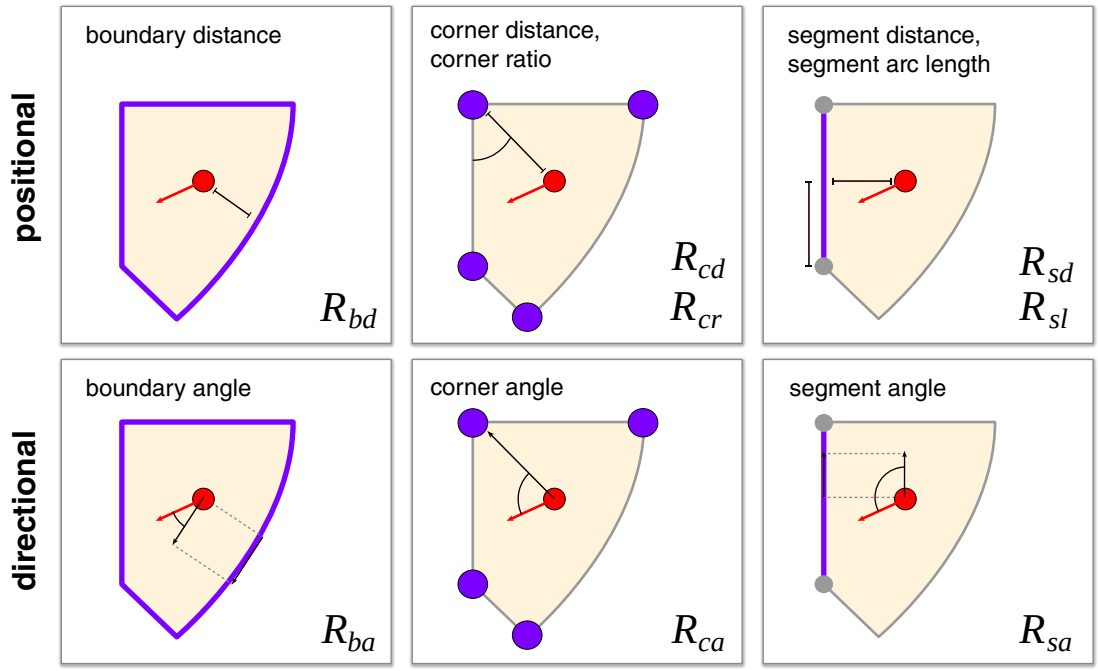


Figure 4.5: Relationship functions between a polygon and a query pose (red). Features (purple) in column from left to right: polygon, corner and segment. The top row shows all positional relationships; the bottom row all directional relationships. One positional and one directional relationship of the same column are used to form pairs of relationships.

- The **corner distance** is the distance between the query point and a polygon corner: $R_{cd}(Q, c_j) = d(Q_p, v_{k_j})$.
- The **corner ratio** is the angle that the direction from corner to query point makes with the start of the first polygon segment adjacent to the corner, normalized with the opening angle of the corner: $R_{cr}(Q, c_j) = \frac{\angle(Q_p - v_{k_j}, v_{k_j-1} - v_{k_j})}{\angle(v_{k_j+1} - v_{k_j}, v_{k_j-1} - v_{k_j})}$.
- The **boundary angle** is the angle between the pose direction and the direction of the polygon boundary at the location of minimum distance to the query point. If $x = \arg\min_{x \in b(A)} d(Q_p, x)$, then $R_{ba}(Q, A) = \angle(\frac{d\, b(A)}{dt}, Q_d)$, where t is a parametrization of the polygon boundary by arc length.
- The **segment angle** $R_{sa}(Q, e)$ is the same as the boundary angle, but considers only a segment $s \in A$.
- The **corner angle** is the angle between the pose direction and the direction from corner to query point: $R_{ca}(Q, c_j) = \angle(Q_p - v_{k_j}, Q_d)$.

The first five relationship functions are called *positional*, because they depend only on the position of the query pose. The other three are called *directional*, as they also depend on the

direction of the query pose. Which of those relationships are actually applied depends on the application scenario, and more general relationships can be used. Later, we will always combine a positional with a directional function to clearly define similar poses. For this purpose we assign one directional relation R_d to each positional relation R_p. For the relationships defined above, we assign each R_{fd}, R_{fr} or R_{fl} the corresponding R_{fa} where $f \in \{b, s, c\}$ according to the type of feature. This set of relationship functions works well in practice, but we can easily remove or add additional functions to the set. For a discussion of the influence of individual relationships on the final result and an example of an additional relationship function, see Section 4.9.

4.4 Pose Matching Importance

In order to find poses in the scene whose neighborhood matches the query pose, we evaluate both a *local importance* and a *matching accuracy*.

Local Importance

The local importance determines how important features near the query pose (i.e., input features) are for determining matching poses. It is therefore again a functional on features and poses:

$$I_l : \mathbf{P} \times \mathbf{F} \to \mathbb{R}.$$

However, while a relationship function is used to determine the *location* of poses with similar geometric relations to the query pose, the importance is used to determine the *influence* each surrounding feature has with respect to a geometric relationship in the final selection of matching poses.

There are many possible importance functions, including semantic weightings assigned by the user, but in our current implementation, we simply use a weight based on the normalized distance to the feature, as defined already by the relationships functions involving distance:

$$I_l(Q, F) = 1 - \min(1, R_{fd}(Q, F)/n(A_F)),$$

where the normalization factor $n(A_F)$ is the radius of the largest circle inscribed in the polygon corresponding to feature F for Q inside the polygon, or twice the radius of the smallest circle enclosing the polygon for Q outside the polygon, and $f \in \{b, s, c\}$ according to the type of feature. This reflects the fact that features close to the query pose should have stronger influence on the matching than features farther away.

Matching Accuracy

The matching accuracy determines whether a “distant” candidate feature for matching (i.e., an output feature) resembles a local feature that has been determined to be important to the query pose. For example, if the query point is positioned near a window-shaped polygon, other window-shaped polygons in the scene will receive higher weighting in the subsequent matching algorithm. The matching accuracy is a functional on features:

$$I_m : \mathbf{F} \times \mathbf{F} \to \mathbb{R}.$$

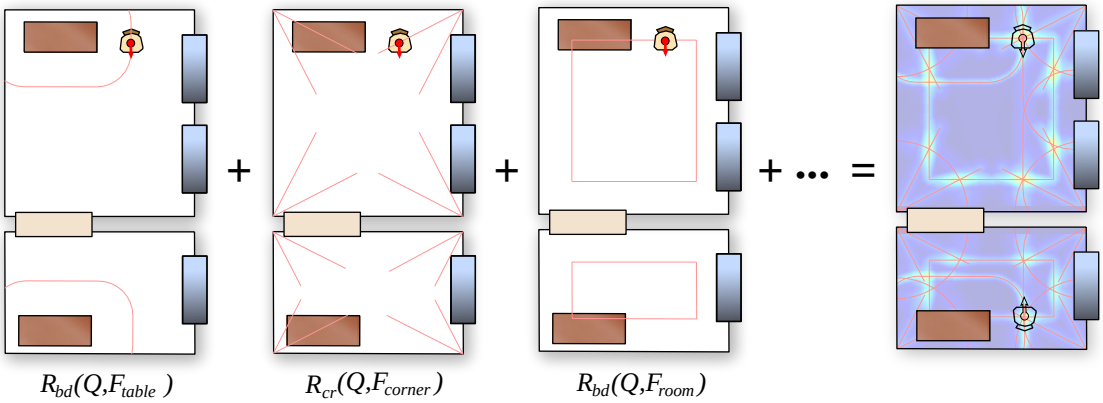


Figure 4.6: Accumulation of level sets (red) of the relationship functions. The level sets of the relationships of the red query pose to local features are accumulated. The maxima of the resulting distribution are poses that have similar function values as the query point. The text below the figure indicates which relationship was used to create the level sets. The image on the right shows the result of the accumulation as heat map.

Obviously, if the two features are not of the same type, a value of 0 should be assigned. In our implementation, we use the following matching accuracies:

- for **polygons**, the similarity of polygon areas: $I_m(A_1, A_2) = \min(a(A_1), a(A_2))/\max(a(A_1), a(A_2))$, where $a(A)$ denotes the area of polygon A.
- for **segments**, the similarity of segment lengths and curvatures:
  $I_m(s_1, s_2) = \frac{\min(t(s_1),t(s_2))}{\max(t(s_1),t(s_2))} * (\max(\tilde{\eta}_1, \tilde{\eta}_2) - \frac{|\tilde{\eta}_1 - \tilde{\eta}_2|}{\max(\tilde{\eta}_1,\tilde{\eta}_2)})$,
  where $t(s_i)$ is the arc length of segment s_i and $\tilde{\eta}_i$ is the average curvature of the segment.
- for **corners**, the similarity of opening angles:
  $I_m(c_1, c_2) = |c_1^\alpha - c_2^\alpha|/\pi$.

Note that there are many possible matching functions, including symmetry-based and semantics-based ones. For example, the matching could be further restricted according to labels assigned to polygons, e.g., matching only features that fulfill some conditions on the labels, as will be shown in Section 4.8.

4.5 Matching Algorithm

Theory

In this section, we discuss the algorithm that computes potential matching poses from the query pose. We assume a set $\mathbf{R}$ of relationship functions to be given, as well as a set $\mathbf{F}$ of features. The main idea is to determine for each triple of relationship function R, "local" feature F_l and

candidate matching feature F_m all poses in the scene that have a similar function value as the query point. In other words, we want to find the **level set**

$$L(R, F_l, F_m) = \{P \in \mathbf{P} | R(P, F_m) = R(Q, F_l)\}.$$

In order to combine the potential candidate poses generated by different relationship functions, local and matching features, we accumulate their level sets in pose space according to local importance and matching accuracy. There are two ways to combine candidate poses: We can either *intersect* different level sets, or we can *accumulate* their importance.

Intersect: We note that level sets of positional relationships $R_p \in \mathbf{R}_p$ are independent of the angle and therefore span the entire angular dimension at each position, while level sets of directional relationships $R_d \in \mathbf{R}_d$ span the entire positional domain. To obtain level sets of finite extent, we intersect the level set of each positional relationship R_p of a pair (F_l, F_m) with the level set of its associated directional relationship $\hat{R}_d$:

$$\hat{L}(R_p, F_l, F_m) = L(R_p, F_l, F_m) \cap L(\hat{R}_d, F_l, F_m).$$

Accumulate: For each pose, we accumulate the importances of the intersected level sets to obtain a final importance I, where, for each relationship, we only take the best match:

$$\begin{aligned} I(P, Q) &= \sum_{F_l \in \mathbf{F}} I_l(Q, F_l) \\ &\quad \sum_{R \in \mathbf{R}_p} \max_{F_m \in \mathbf{F}} \left(I_m(F_l, F_m) \mathbf{1}_{\hat{L}(R, F_l, F_m)}(P) \right), \end{aligned}$$

where $\mathbf{1}_X$ is the characteristic function of set X. The candidate poses we are looking for are then the local maxima of I. Figure 4.6 illustrates this process.

Finding Maxima of I

In practice, the definition of I makes a computation of local maxima computationally unwieldy, since pose space is both continuous and three-dimensional (2D location and 1D direction). In order to solve this problem, we make two simplifications:

- We accumulate importance in 2D, regardless of the directional information of poses, and maintain a direction of maximum importance during the accumulation.
- We discretize the 2D location space.

For this, we first calculate the projection L_p of the level set onto 2D location space:

$$L_p(R, F_l, F_m) = \{P_p | P \in L(R, F_l, F_m)\}$$

and compute the positional importance I_p only in 2D space:

$$\begin{aligned} I_p(P_p, Q) &= \sum_{F_l \in \mathbf{F}} I_l(Q, F_l) \\ &\quad \sum_{R \in \mathbf{R}} \max_{F_m \in \mathbf{F}} \left(I_m(F_l, F_m) \mathbf{1}_{\hat{L}_p(R, F_l, F_m)}(P_p) \right). \end{aligned}$$

For the directional information, we have to employ an approximation. When accumulating in 2D location space, the directional information of the accumulation is lost and we cannot obtain the direction of maximum *accumulated* importance at each position. Instead of accumulating, we use the direction of the level set with maximum importance at any given position. To express this mathematically, we introduce a directional importance I_d defined in full 3D pose space, which, however, is only needed to formally express the result:

$$\begin{aligned} I_d(P, Q) &= \max_{F_l \in \mathbf{F}} I_l(Q, F_l) \\ &\quad \max_{R \in \mathbf{R}_p} \max_{F_m \in \mathbf{F}} \left(I_m(F_l, F_m) \mathbf{1}_{\hat{L}(R, F_l, F_m)}(P) \right), \\ \phi_{max}(P_p) &= \arg\max_{\phi \in [0, \ldots, 2\pi]} I_d((P_p, \phi), Q). \end{aligned}$$

Note that ϕ_{max} may not be the same direction found when accumulating in the full 3D pose space. However, there are only differences in special cases and the 2D approximation does not lead to unintuitive results. For a discussion, see Section 4.9.

Implementation

The evaluation of I_p and ϕ_{max} is carried out on a 2D regular grid. In practice, this corresponds to a *rasterization* of the level sets onto a grid. In all our examples, the grid has approximately $100k$ grid points in total, with the aspect ratio adapted to the extent of the scene. We generate the level sets by rasterizing simple primitives like circles for the corner distance or lines for the segment arc length. For the boundary distance relationship function, we compute a polygon offset by taking the Minkowski sum [6] of the polygon boundary with a circle and then rasterize this polygon offset. In order to avoid aliasing and to increase robustness (e.g., to account for inaccuracies in the input geometry), we replace the binary characteristic function $\mathbf{1}_X$ of the level set by a filter kernel $\mathbf{k}_X$, which slightly dilates the level set. In our examples, we use a linear falloff with a small fixed radius.

The result of this step is a 2D grid with an importance value I_p and a direction ϕ attached to each grid cell. The best candidate locations are then found by applying non-maximum suppression [39] on the importance grid. The resulting maxima are then sorted according to their importance, giving an ordered list of those candidates that best match the query pose.

The level sets are rasterized using line or polygon rasterization in graphics hardware. Each type of relationship function requires a specific method for calculating the level set. For R_{bd}, the boundary distance, for example, the polygon boundary is eroded by that amount that makes the eroded polygon intersect Q_p.

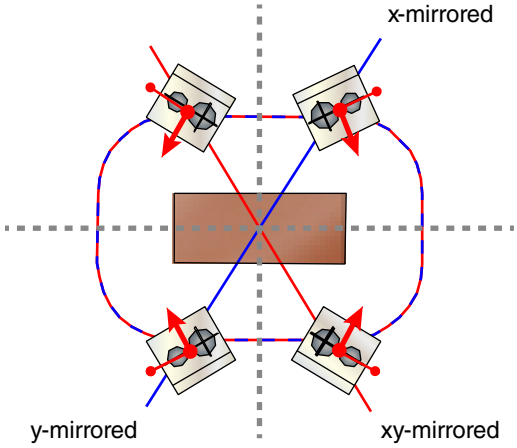


Figure 4.7: Mirrored level sets of an object with two mirror symmetries. When mirroring an object on one of the axes, the resulting object is mirrored. Level sets that correspond to these mirrored poses are shown as blue lines. Non-mirrored level sets are shown in red. The blue/red dashed lines are two level sets with the same position (but different angle), one mirrored and one non-mirrored.

4.6 Mirror Symmetries

Given a query pose Q, it is often desirable to find all objects that have similar relationship function values when *mirrored* along a symmetry axis of a feature. In other words, we want $R(s_{F_m}(P), F_m)$ to give the same result as $R(P, F_m)$, where s_{F_m} is a function that mirrors pose space along one (or more) *symmetry axes* of a feature.

However, if a pose has been mirrored, it does not represent the same object anymore. Instead, it should represent an object that has been mirrored about P_d. In order to be able to represent such mirrored poses, we extend pose space $\mathbf{P}$ to $\mathbf{P}'$ by a binary flag P_m:

$$P' = (P_p, P_d, P_m) \in \mathbf{P}'.$$

To account for mirrored poses, the symmetry operator is extended to handle the mirror flag:

$$s_{F_m}((P_p, P_d, P_m)) = (P_p^m, P_d^m, 1 - P_m),$$

where (P_p^m, P_d^m) is the pose mirrored about a symmetry axis s of F_m. This properly accounts for the fact that after application of two symmetry axes, the pose is in unmirrored space again.

We then construct additional level sets, one for each symmetry axis or combination of symmetry axes s of the matched features (see Figure 4.7):

$$L^s(R, F_l, F_m) = \{P' \in \mathbf{P}' | R(s_{F_m}(P'), F_m) = R(Q, F_l)\}.$$

The remainder of the method is identical: the mirrored and non-mirrored level sets are accumulated in extended pose space $\mathbf{P}'$, and the maxima identify points that have relationship function values similar to the query point, up to symmetries. This is implemented by accumulating importance in two different grids, one for mirrored and one for regular space.

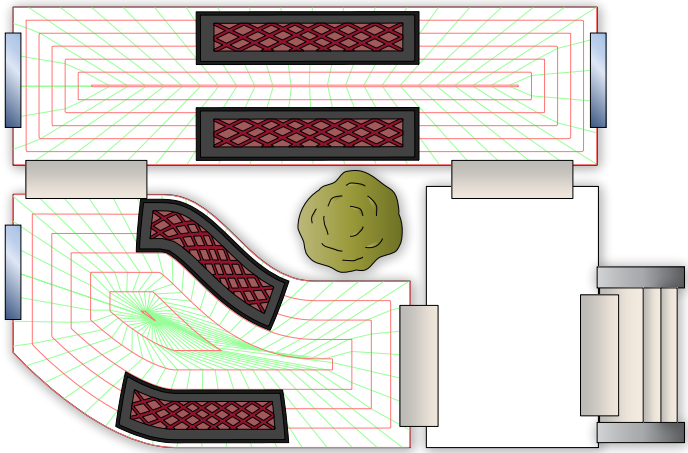


Figure 4.8: Two objects propagated from the inside of the top room to the inside of the bottom room using a local coordinate system based on the boundary distance R_{bd} and a custom relationship R_{bf} based on the arc length from a fixed start point on the boundary to the position on the boundary closest to the query pose. Red lines are the iso-curves for different values of R_{bd} and green lines iso-curves for different values of R_{bf}.

4.7 Local Coordinate Systems

So far, the algorithm works well for placing rigid objects, where considering the pose of the object is sufficient. However, it is difficult to generate a continuous mapping from source to target polygon directly from our method. For example, the map resulting from always taking the highest target maximum for every source pose is generally neither injective nor surjective. In floor plans, it can also be useful to transfer an object *non-rigidly*, i.e., transferring all points from a sample placement to another location in the scene, while preserving both the coherence of the points among themselves and the match of each point to its neighborhood. This requires a more strict definition of neighborhoods in the form of *local coordinate systems*. In this section, we clarify the connection between our relationship functions and local coordinate systems by providing a simple approach for constructing local coordinate systems using our relationship functions. For a more sophisticated approach to establishing a mapping between shapes using local coordinate systems, we refer to Solomon et al. [90].

In a coordinate system, each coordinate can be interpreted as a relationship between the point being described and some geometric feature. Together, the coordinates uniquely specify the point in space. In Euclidean coordinates, for example, each coordinate measures the distance of the point to a plane orthogonal to the coordinate axis. In mean value coordinates, each coordinate is related to a generalized distance of the point to a polygon vertex.

The directional relationships introduced in Section 4.3 can also be used to identify points in space with respect to a feature (i.e., locally). In 2D, exactly two such relationships are required. In order to form a local coordinate system, the selected relationships should provide a bijective mapping from the domain to coordinates. While it is evident that each of the directional

relationship functions cover the domain, the uniqueness of coordinates can be described in terms of the level sets of the relationship functions, which, in the positional domain, correspond to iso-curves:

The mapping of points to coordinates is invertible if any two level sets (L_1, L_2) chosen from the relationship functions (R_1, R_2) intersect at exactly one point, and if R_1, R_2 themselves are injective (i.e., the corresponding iso-curves do not self-intersect).

The local coordinates of a point P_p with respect to a feature F are then simply $(R_1(P_p, F), R_2(P_p, F))$ (see Figure 4.8 for an example). From the relationships introduced in this paper, only the pair (R_{cd}, R_{cr}) fulfills this property. The drawbacks of using local coordinate systems are that the two relations R_1 and R_2 and the feature F have to be determined manually before the propagation and that only coordinates that fulfill the properties described above can be employed. Also, the resulting coordinate system does not necessarily provide a smooth mapping between polygons - there may be some C1 discontinuities. For example, in Figure 4.11, a pattern of flowers has been propagated from one lawn area to other lawn areas using local coordinates (for details refer to Section 4.9).

4.8 Application: Floor-Plan Editing

For specific applications, we can extend our method in various ways. An interesting area of application is floor-plan editing. Manually placing objects in these floor plans can be time-consuming but cannot be trivially automated, since the placed objects have to respect geometric relationships to existing objects. To apply our method to this problem, objects of the floor plan, such as rooms or tables, are modeled as polygons. To place an object, the user defines a pose. We can propagate this pose using geometric relationship functions as described in the previous sections. To specialize our method to this application, we introduce four simple extensions: polygon labels, a hierarchy over polygons that describes inclusion relations, a pose validation step that removes propagated poses that would result in unwanted object intersections, and placing arrays of objects. Additionally, we define probabilistic extensions for object placement.

Labels Since the focus of our work does not lie in object recognition, we assume that all polygons in the floor plan are labeled according to their type, e.g., table, bathroom, and so on. All features F of a polygon have the same label as the polygon. The matching accuracy $I_m(F_1, F_2)$ of two features with different labels is set to zero, reflecting the fact that objects of different types should not be matched.

Hierarchy A floor plan is usually divided into different areas like rooms or buildings. These areas are objects of the floor plan and correspond to polygons in our framework. When placing an object into one of these areas, only relationships to objects in the same area are important. To incorporate this fact into our framework in a general way, we define a hierarchy on all polygons. A polygon A_c is the child of a polygon A_p if A_c is contained in A_p, or if the polygons intersect and A_c has a smaller area. The local importance $I_l(Q, F)$ of a feature is set to zero if Q is not contained in the parent polygon of A_F. This allows, for example, first placing parcels in a larger

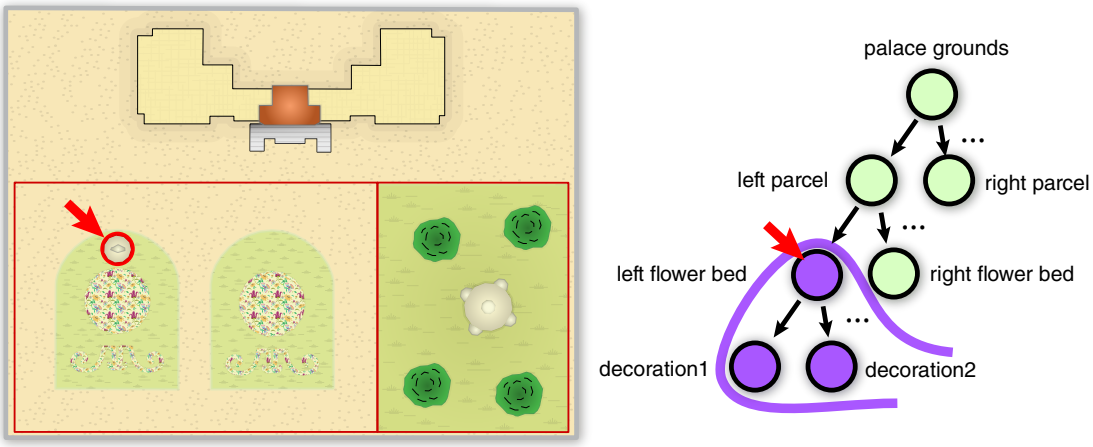


Figure 4.9: The floor-plan hierarchy for a small garden scene. When placing an object in the left flower bed (red arrow), only the siblings and the parent (purple) of the object are considered for I_l.

area, then flower beds in each parcel, and then flowers in a flower bed. In each step, only the parent and siblings of an object will be considered for I_l (see Figure 4.9).

Pose validation The placement algorithm does not take geometric properties of the object (apart from its pose) into account. Therefore, a placed object could have unwanted intersections with the existing geometry. To avoid this, we first collect all labels of polygons that are intersected by the object when placed at the query pose Q and at the candidate pose P. If the labels at P are not a subset of the labels at Q, then the pose is removed. Typically, this allows intersections with the parent polygons at the placed poses. For the intersection step, alternative geometry can be used. For doors, for example, the polygon can be extended to avoid objects being unintentionally placed too close, so that the door cannot be opened.

Arrays of objects We can also facilitate the placement of arrays of objects. An array is a set of copies of the same object that are aligned along a path and have fixed spacing. The path is defined as the offset boundary of a nearby polygon. Arrays can be defined by a start point, an end point, a polygon, and a spacing. The spacing is user-defined and set before the array is created. First, the start point is propagated as usual and all resulting propagated points are marked as starting points. Then, a second point is propagated to mark the end points of the array. The polygon used to align the array is defined as the polygon closest to both start and end points. If the array start and end do not agree on the closest polygon, no array is created. If there are more than two end points on the same boundary, only the end point closest to the start point is used to form the array. For all valid pairs of start and end points, the end point is projected to the offset boundary of the polygon to mark the end of the array along the offset boundary, and objects are placed using the predefined spacing.

Probabilistic pose selection In order to allow more variation, we introduce probabilistic placement of poses. First, we define a scope for probabilistic selection, typically by the parent polygon label (e.g., "room"). Several query poses are defined in the source scope. Then, in each target scope, one of the query poses is randomly selected and only matches corresponding to that query pose are accepted as candidates.

Probabilistic pose placement In Section 4.5, we already introduced a kernel $\mathbf{k}_X$ to avoid aliasing and geometry errors when rasterizing the level sets. This idea can be extended by choosing a wider kernel and replacing the deterministic search for local maxima by a probabilistic one, to allow for more natural (i.e., irregular) placements. This can be done by interpreting the resulting regular grid as a probability distribution and sampling poses according to this distribution.

Parameters In floor-plan editing, we allow for two types of parameters: first, the maximum number of matches allowed for a polygon of a certain label (typically "room") per query pose, and second, a maximum placement density, which can be influenced by changing the radius of the non-maximum suppression.

We use the first three extensions (labels, hierarchy and pose validation) in every operation of our examples. See Section 4.9 for a discussion on the influence of labels on the final result. On the other hand, arrays of objects, probabilistic pose selection and probabilistic pose placement are optional operations that are not necessary for every scene, but give results that would be hard to achieve with the standard propagation operation or further facilitate object placement. All of the operations are used at some point in the examples.

4.9 Results

We have implemented a simple prototype of our method in MATLAB. The prototype has a user interface that supports the operations and settings described in Section 4.8. A typical modeling session starts with a floor plan containing only a few objects, usually only the rooms. Note that it is necessary to have at least one object in the scene before an edit propagation is meaningful. A user of the interface can then place (position, scale and rotate) objects from a library of labeled objects and apply edit propagation. With a slider, the user can interactively set a threshold on the importance of placements that are displayed. The user can then select arbitrary objects and integrate them into the scene. Subsequent edit propagations take relationships to previously placed objects into account.

The computational complexity of an edit operation depends on the number of level sets, which is proportional to the number of features in the source room N_i^S and the total number of features N_i^F having the same label i. Each feature in the source room (or more generally parent object of the source pose) has to be matched with all the features having the same label in all target rooms, resulting in a computational complexity of $O(\sum_{i \in L} N_i^S N_i^F)$, where L is the set of labels in the scene. Each level set is rasterized on a 2D grid using standard rasterization techniques, as described in Section 4.5. In practice, N_i^S does not depend on the complexity of the scene, since,

for example, larger floor plans usually have more rooms, but not more objects inside a room. N_i^S is also typically very small compared to N_i^F, so the complexity is approximately linear in the total number of features of a scene. In our implementation, one edit propagation typically takes between 1 and 6 seconds, up to approximately 15 seconds in the fully populated crown example (Figure 4.12). This example is close to worst-case for our method, since all features are placed in only two 'rooms' (i.e., the two crowns), resulting in a large N_i^S. Please note that our algorithm is an unoptimized proof-of-concept, and timings can be improved.

We demonstrate our method by populating two extensive scenes with a large number of objects. The goal is to create plausible object positions with few edit operations. We show only the result and the starting configuration for each of the two scenes. Unless noted otherwise, all relationship functions described in Section 4.3 were used for each edit operation. In a typical editing session, we placed the objects in no particular order, only following the semantic dependencies of objects (e.g., placing the night tables after the bed; see Section 10.1 for a discussion on the order of operations). A few operations required one or two retries with small adjustments of the input pose, e.g., if there was too little free space in one of the target rooms to fulfill the requested geometric relationships.

The first scene is one floor of an apartment building, as shown in Figure 4.10. The starting configuration for this example is a bare room layout. Rooms are labeled as bedrooms, living rooms, kitchens, closets, toilet rooms, and bathrooms. For illustration, different apartments are shown in different colors, although this has no effect on the algorithm. Several pieces of furniture are propagated successively, as shown by the red numbers. The manually placed query object for each operation is encircled. Note that even though the rooms have different shapes and sizes, and the furniture inside the rooms has different arrangements, our method ensures that object positions are always plausible and similar to the position of the query object. In this example, a total of 160 objects were placed with 26 edit operations.

The second scene is a palace garden, shown in Figure 4.11. Here, the domain is divided into several parcels, and the starting configuration only includes the boundaries of these parcels. Parcels are labeled as way, garden, and palace. These parcels only serve as a guide for the placement of the actual objects and are deleted at the end of the editing session. Placed objects include flower beds, flower decorations inside the flower beds, fountains, trees, statues and pagodas. In this scene, we use several different operations to propagate objects. In addition to the usual edit propagation, we use object array operations to place the objects that are aligned in straight lines or circles, such as the bushes around the fountains. In step 19, we use local coordinate systems defined with respect to the selected flower beds to propagate all flower decorations of the source flower bed in one operation. The local coordinate system is composed of the boundary distance relationship R_{bd} and a custom relationship R_{bf} based on the arc length from a fixed start point on the boundary to the position on the boundary closest to the query pose. To propagate the large trees, we applied the probabilistic pose placement described in Section 4.8. A total of 617 objects were placed with 27 edit operations.

Our method is not limited to architectural applications, but it can also handle other types of polygonal scenes as long as it is possible to find a clear set of relevant features (we discuss suitable types of scenes in Section 4.9 and Figure 4.20). In Figure 4.12, we demonstrate placing gems and ornaments on two crowns. Since we only handle two-dimensional domains, the crowns

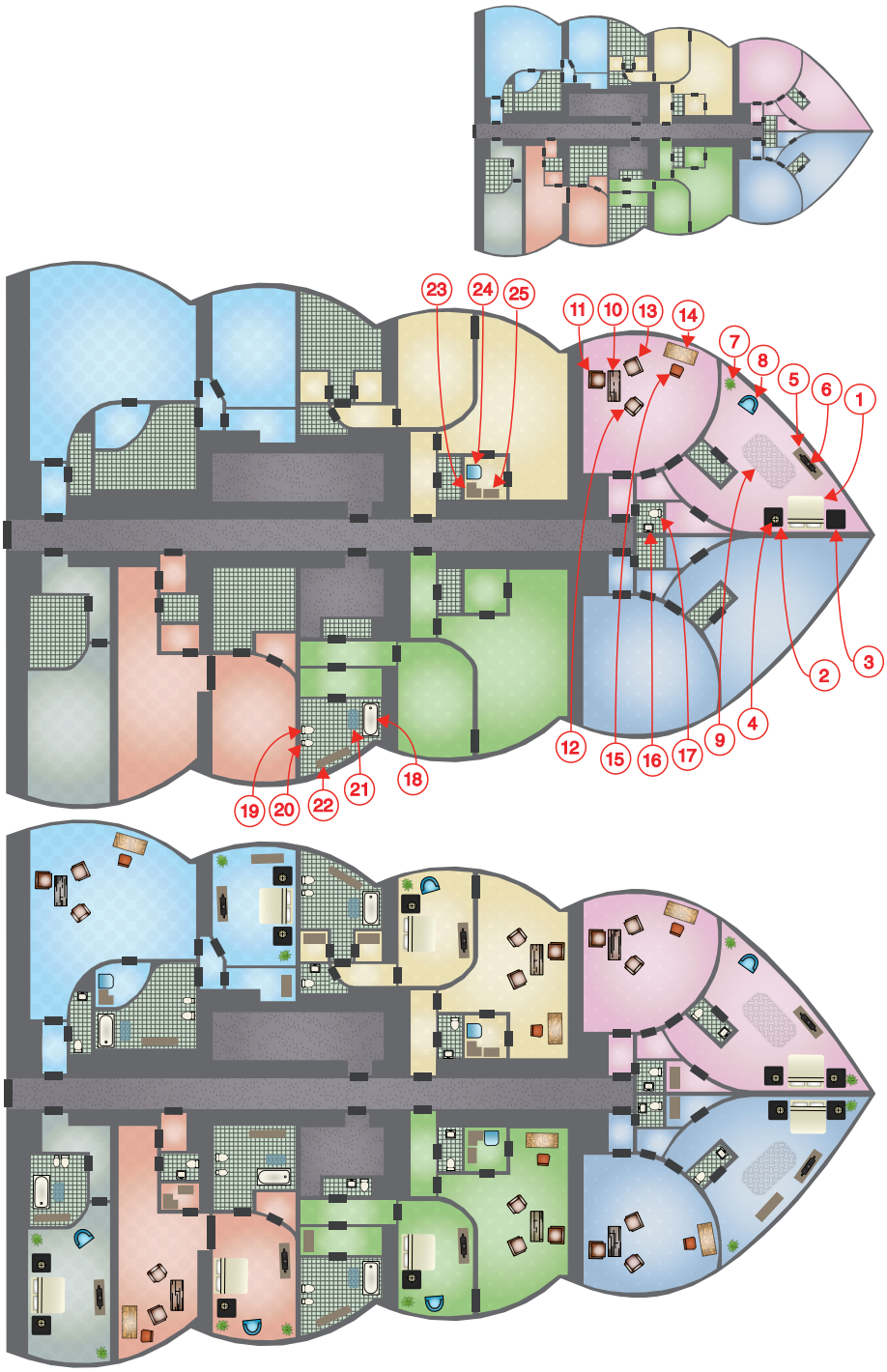


Figure 4.10: In the apartment-building scene, 160 objects were placed with 26 edit operations. The top image shows the starting configuration, the middle image shows the individual edit operations – numbers indicate the order of the operations – and the bottom image shows the final result. Zoom in to see details.

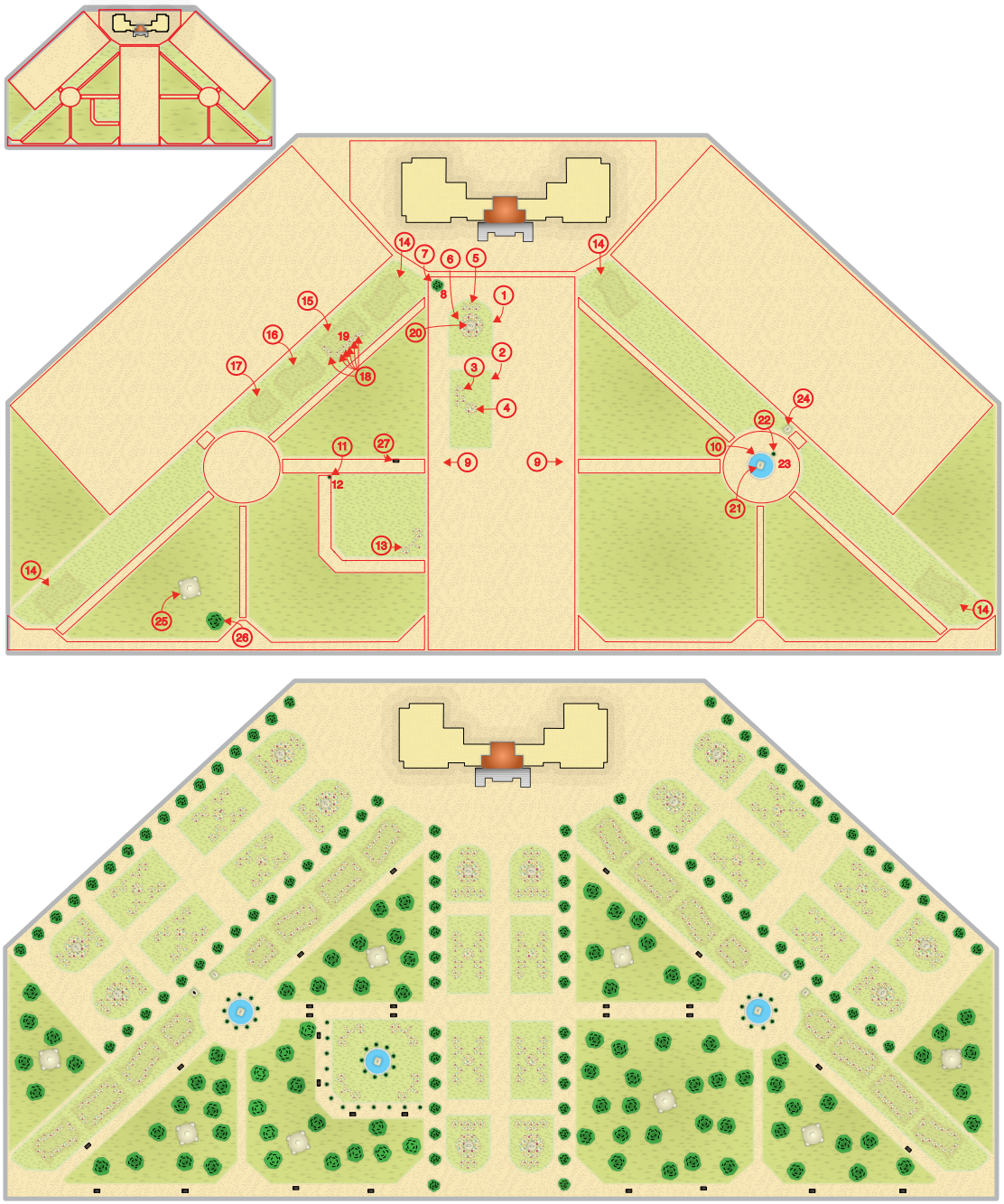


Figure 4.11: In the palace-garden scene, 617 objects were placed with 27 edit operations. The top image shows the starting configuration, the middle image shows the individual edit operations – the numbers indicate the order of the operations – and the bottom image shows the final result. The red lines in the top and center image are polygons that guide the initial object placement and are removed in the end. Zoom in to see details.

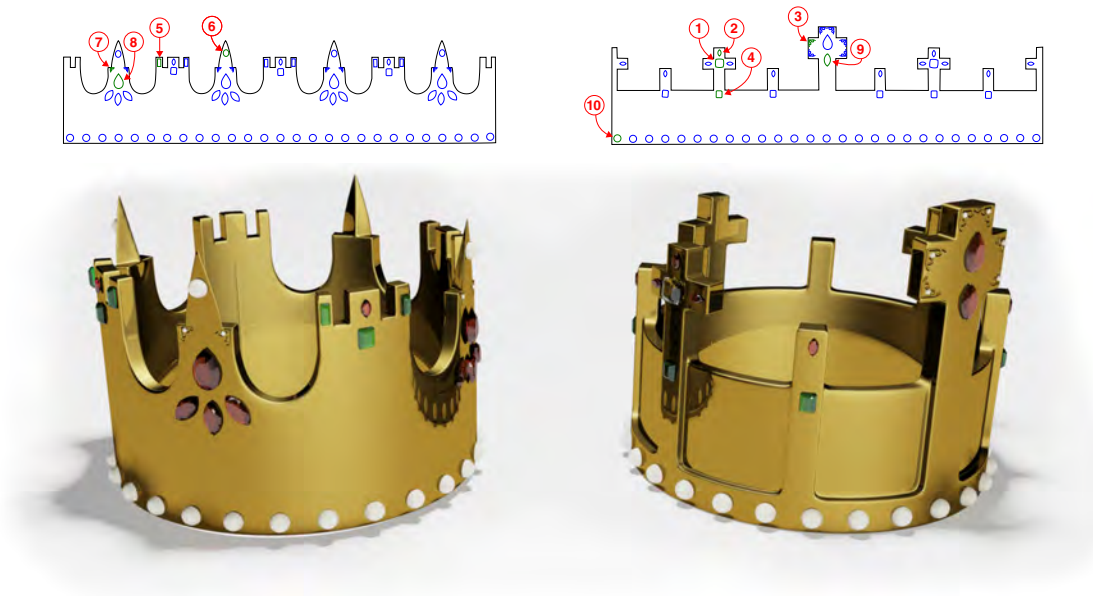


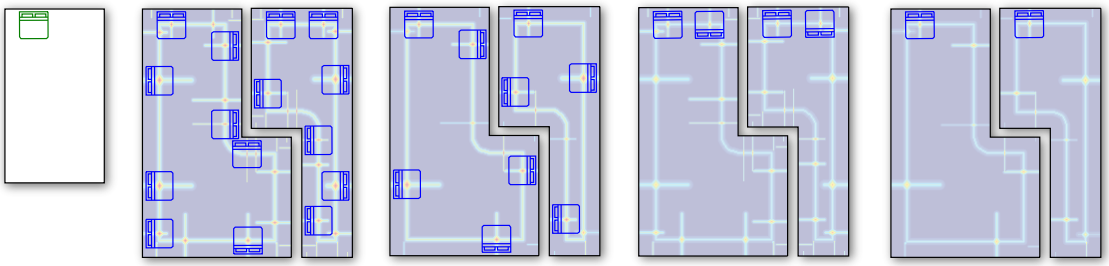
Figure 4.12: Placing gems and ornaments on two crowns. In this example, we place all gems and ornaments for both crowns in ten edit operations. Each edit operation is propagated to both crowns. The crowns are unrolled using a cylindrical mapping before performing the edit operations (top row). Source objects are shown in green; propagated objects are shown in blue. The bottom row shows the final result mapped back to the crowns.

are unrolled using a cylindrical mapping before performing the edit operations (Figure 4.12, top row). The starting configuration is the outline of the two crowns shown in black. Objects labeled as ruby, emerald, pearl, diamond and ornament are propagated on the crowns. Note that each edit operation is propagated to both crowns. In each operation of this example, we use the *centroid direction* relationship function in addition to the other relationship functions. The centroid direction is the angle between the direction from pose to object centroid and the principal direction of the object. As mentioned earlier, our method can handle arbitrary relationship functions. In step 10, we use an object array operation to place both lines of pearls. A total of 122 objects were placed in 10 edit operations. Finally, we map the result back to the 3D crowns (Figure 4.12, bottom).

Discussion and Limitations

In the following, we discuss several aspects of our method. We clarify the influence of the individual terms on the final result, examine approximations used in our approach, provide comparisons to shape matching with a local descriptor and discuss several limitations of our method.

Influence of Local Importance and Matching Accuracy We use two terms to determine the relevance of a level set: the local importance I_l and the matching accuracy I_m. Both are needed



source room | (a) no importance, no matching accuracy | (b) importance only | (c) matching accuracy only | (d) importance and matching accuracy

Figure 4.13: Influence of the local importance I_l and the matching accuracy I_m on object placement in a simple scene. By setting I_l to a constant (a and c), all features in the source room are weighted equally, independent of the distance to the source pose. When I_m is a constant (a and b), all features of the target room have the same matching quality. Low-quality poses in the target rooms are filtered out only when using both terms (d).

to filter out level sets of low relevance. We show the effect of setting either term to a constant in Figure 4.13. For clarity, we use only the boundary distance and segment arclength relationship functions in this example. When setting I_l to a constant (Figure 4.13 a and c), all features in the source room have the same importance, e.g., the segment on the far side of the room has the same importance as the segments next to the bed. The level sets from these less-important features create maxima of low quality, such as the beds facing the wrong direction in Figure 4.13 c. By setting I_m to a constant, all features have the same matching quality, e.g., a match between a long and a short segment has the same score as a match between two short segments. The results are additional maxima along the badly matched segments. Undesirable poses are only filtered out when using both terms (Figure 4.13 d).

Influence of Relationship Functions Another interesting aspect to discuss is the contribution of each relationship function to the final object placement. Each contribution varies from operation to operation and between output poses of the same operation. The relative contributions depend on the types of level sets that contribute to each maximum in pose space. Figure 4.14 shows the contribution of each relationship function in a typical sequence of operations. Level sets are shown in red. The composition of each maximum belonging to either the green source objects or the blue output objects is shown in the histograms next to the rooms. For example, the placement of the bath tub in the first operation depends mainly on boundary distance, segment arc length and segment distance. In many cases, not all relationships present in the source pose (green) are also present the output poses (blue). For instance, the bathtub in the lower-right room does not have the same segment arc length as the bathtub in the source room. Although not all relationship types are used in every edit operation, none of them is superfluous. Each relationship type is used at some point in the whole sequence of operations. Note that as in the crowns example, in this example we also use the 'centroid direction' relationship type.

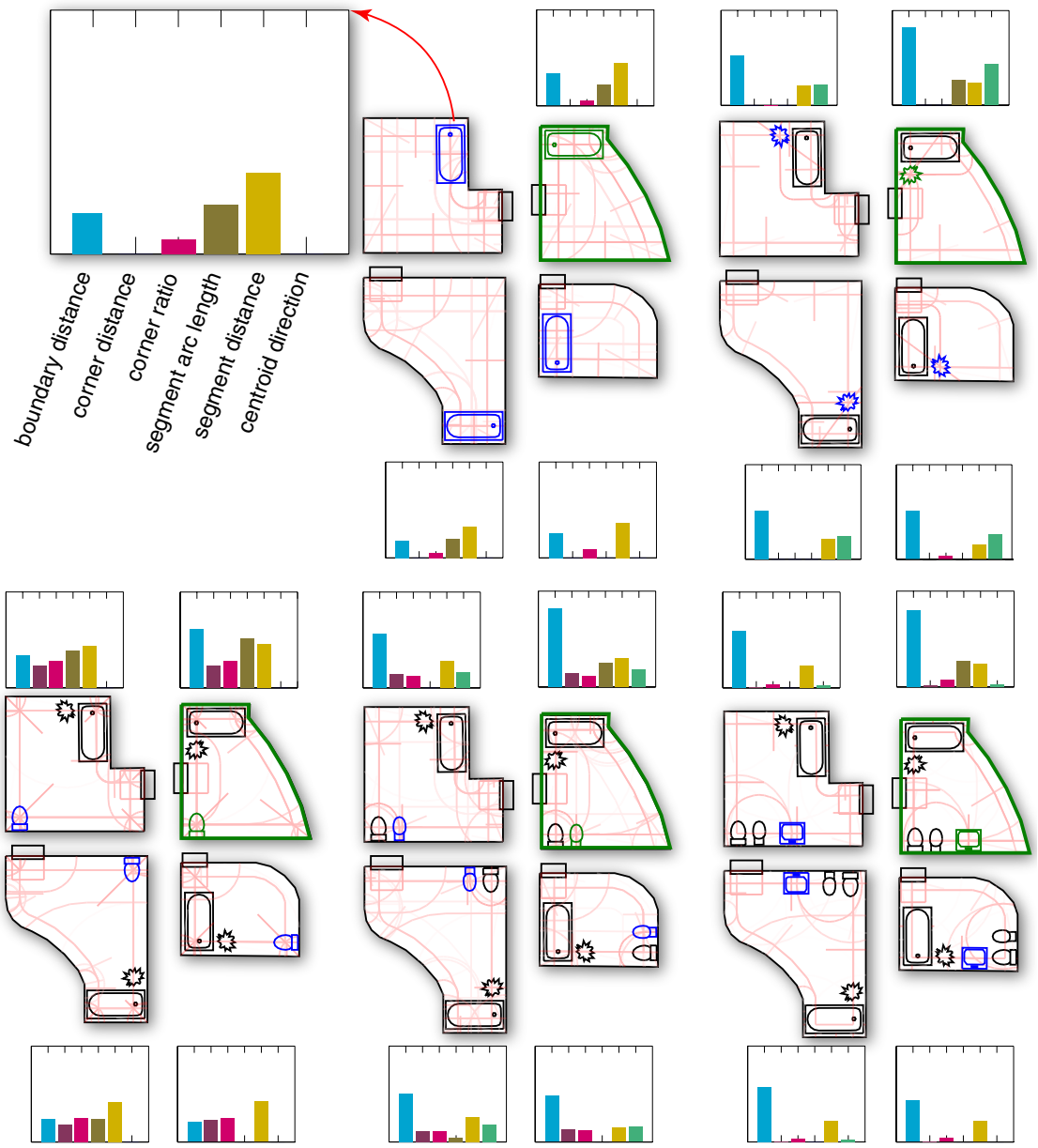


Figure 4.14: Influence of the individual relationship functions on object placement. The histogram bars show the relative influence of each relationship function on the propagated poses, as well as the importance of each relationship type for the source pose in the green room. Level sets are shown as red lines in the background. Note that in general not all relationships are used to propagate a pose, but all relationships are used at some point in the sequence of operations.

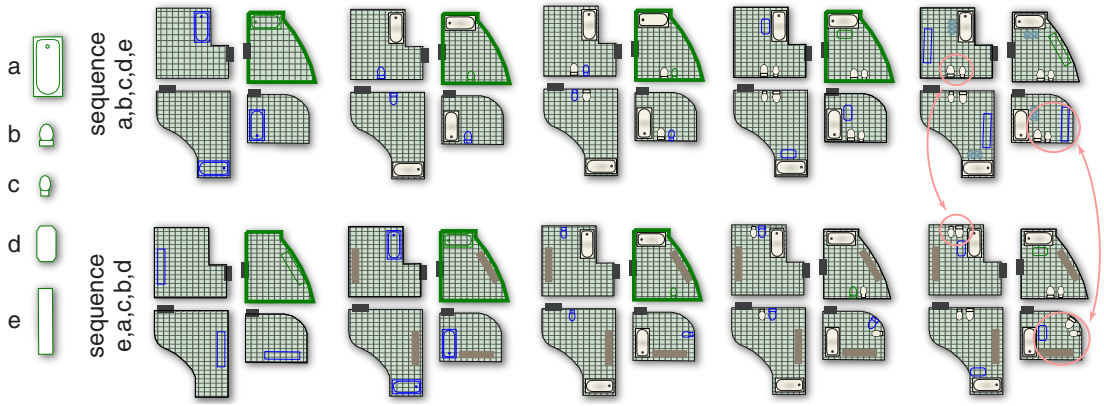


Figure 4.15: Dependence on the order of operations. When propagating the five objects, bathtub (a), toilet (b), bidet (c), bathmat (d) and shelf (e), from the green source room to three target rooms, the final result depends on the order of operations. The top row shows each step of operation sequence a,b,c,d,e; the bottom row each step of sequence e,a,c,b,d. Note how object placement depends on the order of operations. Different placements are encircled in the right image.

Order Dependency When propagating multiple objects, the final result usually depends on the order of operations. Objects are propagated one at a time, and each propagation takes objects that have already been placed into account. Figure 4.15 shows the result of propagating five objects using the same operations, but in two different orders. Note how the shelf placed in the first operation of sequence e,a,c,b,d is propagated to a different pose in the lower right room than in the sequence a,b,c,d,e (last operation), since there are fewer relationships that constrain its placement. Both sequences result in different but plausible object placement. Often there is a natural dependency between objects that suggests a certain order of operations. For example, the placement of the bathmat depends on the placement of the bathtub and should therefore be placed after the bathtub. If there are no such constraints, it is generally advisable to place larger objects first, when there is still enough free space for them. In future work, we may consider transferring all objects in a room at once and automatically resolving all mutual constraints between the objects. However, in many cases, incremental furniture placement might still be preferable, since the amount of work is the same and the user has more feedback as well as more control over the final result.

Influence of Labels In our floor-plan editing application, we rely on labels to provide additional information about objects in a scene. In actual floor plans, relationships often depend on the semantics of objects in addition to their geometric relationship. For example, a night table is usually found next to a bed and not in the living room. Since the focus of our method does not lie in object recognition, we use labels to capture the semantics of objects. Figure 4.16 compares the results of propagating a night table in a typical floor plan without labels (left) and with labels (right). Note that without labels, there are additional maxima in the living room because our method has no way of distinguishing between beds, tables and chairs or between living rooms and bed rooms. However, the method does not break down completely. Even though the results are

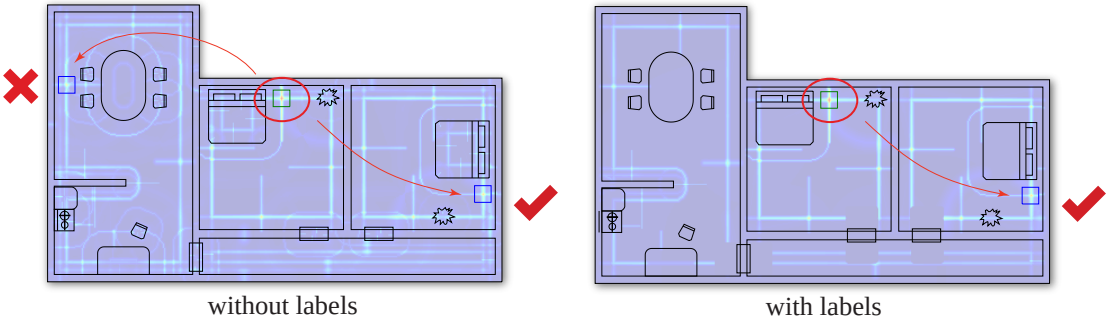


Figure 4.16: The effect of labels on object placement. We propagate the single night table encircled in red. Note that without labels, there are additional maxima at positions that have relationship values similar to the source pose, but to object types that are not present in the source room. The accumulated level sets are shown in the background.

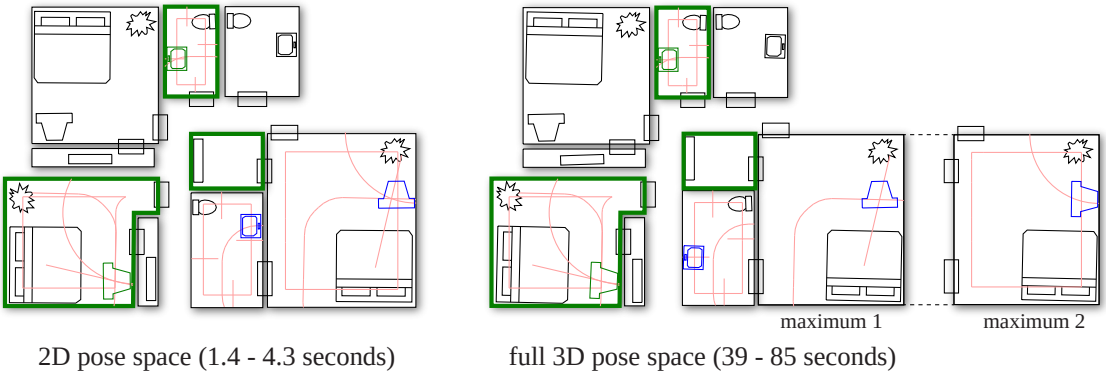


Figure 4.17: Level set accumulation in the 2D space of positions (left) as described in Section 4.5 and in full 3D pose space (right). We propagated all objects in the rooms marked in green (bedroom, toilet and closet) in six edit operations to the remaining rooms. Note that the result is mostly identical except for small differences introduced by the propagation of the two objects shown in green (TV and lavatory). Relevant level sets of the two operations are shown as red lines in the background. For the TV propagation in full 3D pose space, there are two equivalent output poses at the same position, shown as two versions of the same room on the right.

unintuitive for a floor plan, geometric relationships (e.g., close to the inner boundary of a polygon, close to other polygons from the outside, approximately at the center of a polygon segment) are met by all propagated night tables.

Approximation of 3D Accumulation When accumulating level sets, we approximate the full 3D pose-space accumulation (position and direction) with a more efficient accumulation in 2D position space and take the direction with maximum weight at each position, as detailed in Section 4.5. For most cases, this results in the same output poses as in the full 3D pose-space accumulation, but there are some cases where the results are different. Figure 4.17 shows the same sequence of operations performed on the same scene with standard accumulation in 2D

position space (left) and accumulation in full 3D pose space (right). All objects in the green source rooms are propagated to the remaining rooms in six edit operations. As in the crowns example, in this example we also use the 'centroid direction' relationship function. Note that most objects are propagated to the same poses (the small differences in direction are due to the discretization of the direction in the full 3D pose space). Two differences are introduced during the propagation of the two blue objects.

The first difference is the placement of the TV. In the source room, the TV is parallel to the wall and facing the bed. These two directional relationships cannot be satisfied in the far right room – the TV must either be parallel to the wall or facing the bed. In this room, there are two level sets with directions parallel to the wall (corner distance and boundary distance from wall) and two level sets with directions facing the bed (boundary distance and centroid direction from bed) meeting at one position. In the 2D case, all four level sets form one maximum and the direction with maximum weight is taken (facing the bed). In the 3D case, there are two maxima, shown as two versions of the same room in Figure 4.17. One maximum has a direction facing the bed; the other one is parallel to the wall. The four level sets do not actually meet in full 3D pose space, only the level sets shown in each of the two versions of the room meet. When using the full 3D pose space in practice, we would have to resolve such overlapping objects by either letting the user pick one of the objects or by automatically taking the object with maximum importance.

The second difference is the lavatory placement in the lower toilet room. Similar to the TV placement described above, two level sets with different directions (boundary distance to door and boundary distance to wall) are accumulated in the 2D case, whereas in the 3D case, the two level sets do not meet. Unlike the TV propagation, there is no maximum at the same position (no level sets that meet at this position) and a different maximum is chosen instead. As a result of the different lavatory placement, the toilet is paced at a different position as well. Note that in most cases, the result of 2D position space and full 3D pose-space accumulation are the same and none of the differences result in unintuitive object placement. However, the speed gain and memory saving from accumulating in 2D position space are significant. Using 2D position space is more than one order of magnitude faster (1.4-4.3 seconds per operation in 2D position space versus 39-85 seconds per operation in full 3D pose space) and needs almost two orders of magnitude less memory (9.1 MB for discretizing 2D position space versus 762.9 MB for discretizing the full 3D pose space – in this example, directions are discretized into 100 bins).

Comparison to Shape Contexts Local shape descriptors like the popular shape contexts [7] can be used to establish a mapping between two polygons. In Figure 4.18, we compare our method to two methods based on shape contexts. A single object is propagated from the source room to three target rooms (rows 1-3) and from a room containing multiple objects to a target room containing a different arrangement of objects (row 4).

The first method (SC B-Spline), based on code by Dirk-Jan Kroon [54], establishes a one-to-one mapping between the source room and each target room. Shape contexts are computed at regularly sampled positions on the boundary of source and target rooms, and matching pairs of shape contexts provide constraints for a B-Spline deformation grid. The grid is refined iteratively to be as smooth as possible and to avoid fold-overs. In the case of multiple objects, shape contexts are placed on the boundaries of all objects. Affine transformations of the source objects (first

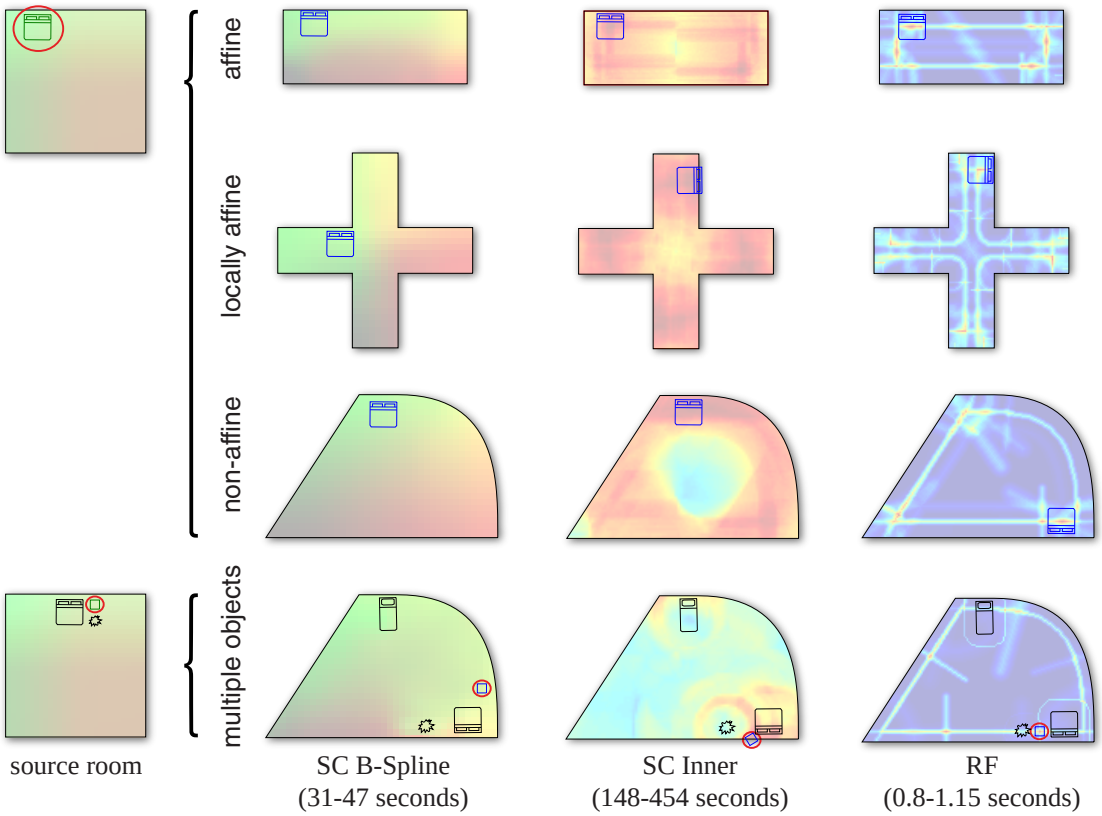


Figure 4.18: Comparison to Shape Contexts. We propagate a single object from the source room to three target rooms using Shape Contexts with a B-Spline deformation grid (SC B-Spline), Shape Contexts computed at grid points inside the polygons (SC Inner) and our Relationship Functions (RF). The first room (top row) is an affine transformation of the source room; the second room (second row) is locally affine to the source room; while the third room (third row) is neither. The last row shows the result of propagating a night table from a source room containing multiple objects. The SC B-Spline column is a one-to-one matching, so we show the deformed coordinates of the source room color coded in red and green. In the other columns, we show the accumulated level sets as heat maps. Note that in the affine row, all methods give good results. As the transformation from source to target shapes becomes less affine, the maxima of the Shape Context become less pronounced. Our method finds good poses and clear maxima in all four cases, since it does not rely on partial shape matching where the matches have to be related by an affine transform.

column) are handled well by shape contexts, but stretching occurs for rooms that are not related by an affine transform (second column), resulting in bad object placement. Since the mapping from source to target room has to be kept diffeomorphic, the mapping cannot accommodate all matches between multiple objects in source and target rooms (last row) if they do not have the same spatial arrangement.

The second method (SC Inner) is a straightforward application of shape contexts to the area of the polygon (as opposed to the boundary). We construct a three-dimensional regular grid over all poses (position and direction) inside the source room. A shape context is computed at each

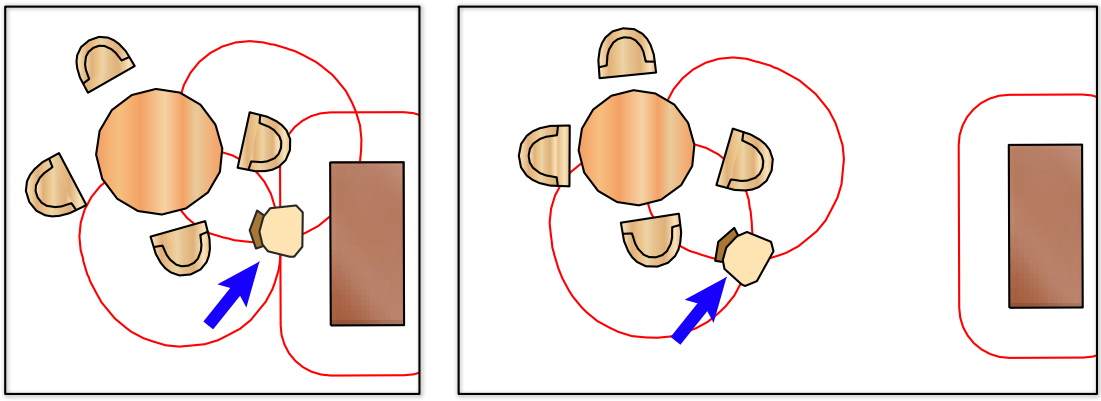


Figure 4.19: Our method cannot handle cases where the desired local importance I_l does not depend on proximity. The red level sets show that the propagation uses the geometric relationships to the chairs instead of the desk to do the propagation. This produces an unintuitive result.

grid point and compared to the shape context at the source pose. The quality of the best match at each position is shown in the background of each target room. The bed is placed at the pose with highest matching quality. Since shape contexts cannot match geometry related by a non-affine transform, no clear maxima can be found in the second, third and fourth rows of Figure 4.18 for the 'SC Inner' method. Our method, on the other hand, is neither constrained by a one-to-one mapping, nor does it require the source and target geometry to be related by an affine transform. It can find clear maxima and good positions in all of the rooms.

Distance-Based Local Importance Currently, the local importance I_l of relationships is determined by a predefined equation based on distance to the source pose. In some situations, however, it might be preferable to base the importance on other criteria. Consider Figure 4.19, where the chair marked by the arrow is propagated from the left to the right room. The only relevant relationship for the chair is to the writing desk. The relationships to the chairs around the other table are irrelevant. However, as is illustrated with red level sets, the relationship to the two chairs is stronger and produces a higher maximum, resulting in an unintuitive placement for the chair. Similarly, scenes might be constructed where distant features are more important than nearby features. These situations can currently not be handled by our method, but we plan to explore strategies for learning I_l from user interactions in future work.

Increasing Scene Difficulty Since our approach focuses more on the relationships of a pose to features than on the features themselves, our method favors scenes in which the quality of a propagated pose is defined mainly by relationships rather than matching quality. The similarity of two features is determined only coarsely, using high-level information such as the polygon area or segment arc length (see Section 4.4). For this reason, scenes with many simple features are best suited for our method (see Figure 4.20). Decreasing the number of relevant features while increasing their complexity reduces the effectiveness of our approach, and would make it necessary to incorporate more selective feature-matching algorithms. Although we could

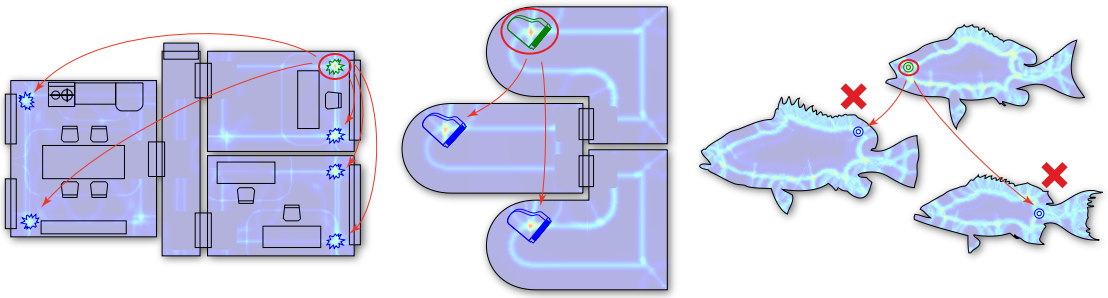


Figure 4.20: Suitable types of geometry. We show three scenes with geometries that are increasingly difficult to handle for our approach. The left scene is well suited to our method; in the middle scene, we can still find good poses, while the right scene is not suitable. Since our method is based on relationships to features, not on feature matching, decreasing the amount of relevant features while increasing their complexity reduces the effectiveness of our approach.

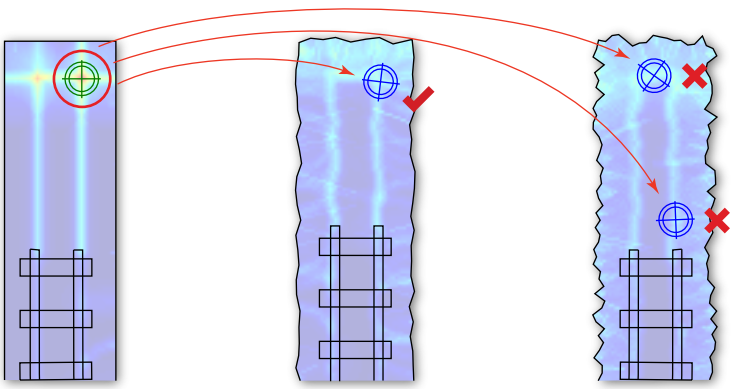


Figure 4.21: We show three shapes with increasing noise from left to right. A single object encircled in red is propagated from the left shape to the two shapes on the right. Since we make no attempt to de-noise the shapes, noise can introduce false features like additional corners and segments that clutter pose space and reduce the propagation quality (middle shape) or even completely hinder propagation (right shape).

use more sophisticated feature-matching methods to compute the matching accuracy I_m, e.g., methods based on the local neighbourhood of a feature, this is not the focus of our method and we opted to keep the computation of I_m simple.

Noise Similar to overly complex local structures, noise in the input shape influences the usefulness of our method. In our current implementation, we make no attempt to remove it. Consequently, adding noise to the boundary of shapes effectively increases their complexity and introduces many false features like additional corners and edges (see Figure 4.21). As explained in the last example, increasing the complexity of features decreases the efficiency of our method. Small amounts of noise can be handled (Figure 4.21, center), but at higher noise levels, false

features become more pronounced and clutter pose space, thereby obstructing good maxima (Figure 4.21, right). In future work, we plan to remove the noise in a pre-processing step, e.g., by adapting an L1 reconstruction method [2] or using bilateral mesh denoising [30] to preserve sharp features.

4.10 Summary

In this chapter, we have introduced the notion of geometric relationships for edit propagation in polygonal scenes. We have exploited the fact that for propagating object placements, mostly the object *pose* with respect to neighboring objects is relevant. Geometric relationships are more flexible and general than local coordinate systems, and we have shown that a local coordinate system can be constructed from certain relationships as a special case. The similarity of geometric relationships is a novel type of geometric analogy that can be used to find similar poses in shape arrangements that have a low degree of similarity. The method is suitable for propagating object placements in general polygonal environments, and we have shown an application to floor-plan editing, where it is especially useful when populating large layouts.

The advantage over knowledge-based algorithms is that no semantic knowledge about the environment is necessary, and since placements are specified by example, they can easily adapt to many different styles of floor plans. Compared to example-based methods that provide complete scenes as example, our method provides more direct user control; feedback is provided after each operation, and the user can adapt operations based on this feedback. Additionally, our method only requires a single example pose, no large database of example scenes is necessary.

CHAPTER 5

Multi-Example Geometric Analogies for Shape Arrangements

Figure 5.1: New York-style skyscrapers generated with the method presented in this chapter. Individual operations were learned in an example editing session. New skyscraper variations can be generated without user interaction.

5.1 Introduction

When modeling a complex three-dimensional scene like a city, several seemingly different modeling operations have to be performed: streets, parcels and building footprints have to be specified; the coarse shapes of buildings, their facades, and finally their interiors have to be

modeled. An important question is then if it is possible to use a single modeling paradigm to facilitate all of these operations. One recurring task in these modeling steps is to place shapes on a region, like windows and doors on a building facade, different buildings on a street layout, or furniture in a floor plan. In this chapter, we show that propagating these placements can facilitate several different modeling operations. An algorithm can analyze the neighborhood of given example shapes and search for qualitatively similar locations in the scene.

In the previous chapter, we presented an example-based editing system that computes the *geometric relations* of a single oriented example point to the rest of a 2D polygonal scene and then propagates the point to locations that are geometrically analogous to the example: locations that exhibit a similar set of geometric relations to the surrounding geometry. In that system, a user can only provide a *single example placement*, which is often not enough to uniquely specify the desired properties of shape placements. For example, a single placement is ambiguous when transferring a shape from a large rectangle to a small one: should the shape scale with the rectangle or keep the same size? Should it keep the same distance to an edge of the rectangle or to its center? Specifying multiple examples in rectangles of different size disambiguates the placement. Furthermore, representing shapes by a *single point* limits the class of relations that can be captured. For example, a user may want to place a balcony on a facade above a door and in front of two windows. In this placement, different parts of the balcony are constrained by different relations: the bottom part of the balcony needs to be close to a door, while the sides need to be close to two different windows. Placements of this type cannot be captured when representing shapes with a single point. Finally, only strictly 2D layout problems are considered.

In this chapter, we present a method to learn the placement of *simple shapes* (as opposed to mere points) in 2D polygonal scenes from a set of *multiple example placements* given by the user. We use the term *placement* to refer to the position, width and height of a shape. A model of the supposed user intent is built using the kernel regression method, for which we introduce a new distance metric between shape placements that works even if the placements have different numbers and types of geometric relationships. We prove that the kernel based on this distance metric fulfills the validity requirements for kernel regression. Furthermore, we show how the 2D placement method can be extended for 3D scenes using extrusion and Boolean operators. We show the application of our method in different domains, including placing elements in city layouts, on building facades, and on interior floor plans, demonstrating cases where previous methods fail. By recording user interactions and applying them to larger scenes under random variations, our example-based shape placement method can be used for more general modeling tasks as well, for example creating multiple variations of a modeled building.

5.2 Overview

The input to our method is a polygonal scene $\mathbf{A} = A_1, \ldots, A_n$, composed of a set of simple polygons A_i, and a set of example *placements* of a shape S given by the user. S is a polygon, but we only consider the axis-aligned bounding boxes $\mathbf{B} = B_1, \ldots, B_m$ of the example placements to compute new placements, therefore a placement consists of four free parameters of the bounding box: the x and y-coordinates of the center position, width and height. Given these inputs, we learn a set of placement rules that allow us to find similar placements in the same polygonal

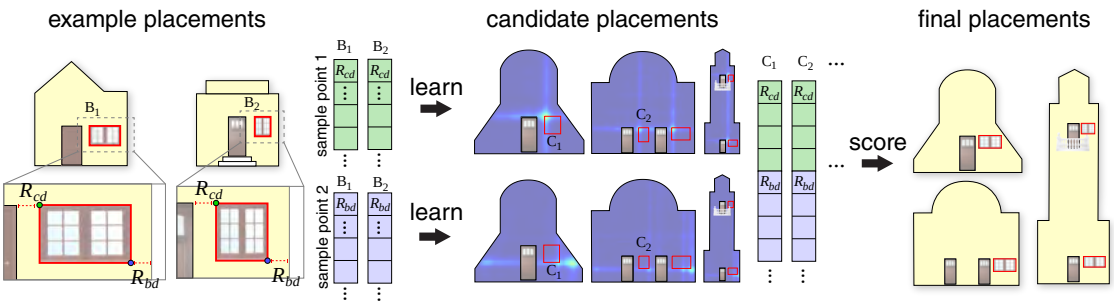


Figure 5.2: The main steps of our method. Given a set of example placements, shown on the left in red, our method samples the placement bounding box sparsely (shown in green and blue) and computes a feature vector containing relationships to relevant scene elements for each sample, denoted in this example by R_{cd} and R_{bd}. These feature vectors are used to train n statistical models, where n is the number of sample points per placement. The models predict good new positions on a dense grid of points, shown as heat map in the center. Candidate placements are constructed from pairs of maxima and scored using a combined statistical model for all sample points.

scene, or in other polygonal scenes that may be much larger than the original scene, without the need for further user input.

Intuitively, we want to learn the user intent from the example placements. In the case of city modeling, a user intent may be *"20 cm below each window"* on a building façade, or something less concrete like *"close to a house but not close to a street"* in a city layout. In many cases, a single example placement is too ambiguous to clearly reveal the intention of the user. A set of examples helps narrowing down the user intent.

Our notion of placement similarity is based on the geometric relationship of the bounding boxes to the surrounding geometry, described by a feature vector. We use *relationship functions*, as described in Chapter 4, such as the distance to a polygon boundary or the direction to a polygon corner, to compute the entries of a feature vector.

Two placements are similar if the distance (with respect to an appropriately chosen distance function) between their feature vectors is small. We use a non-Euclidean distance to compute the difference between feature vectors. The choice of distance function can be motivated by two observations: First, the exact value of the distance between two individual feature values is usually not relevant, it is only relevant whether the features agree (within a set tolerance) or not, e.g., it is irrelevant if a boundary distance at a point has a value of 100 or 1000 units if the desired value for the boundary distance is 1. Second, we use a large set of features that can model a large range of user intents. For a given input s_i, this set of features over-represents the user intent, i.e., two similar placements typically only agree in a small subset of the features. Therefore, a single unmatched feature should only have a limited contribution to the distance.

The smoothed ℓ_0 (SL0) distance addresses both of this issues. It counts the number of features that disagree in their values instead of penalizing unmatched features based on the difference in values and is defined as [78, 76, 23, 96, 99]:

$$d(\mathbf{x}, \mathbf{y}) = D - \sum_{k=1}^{D} \mathcal{G}(x_k - y_k | 0, \sigma), \tag{5.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two feature vectors, D is the dimension of the feature space and $\mathcal{G}(x|\mu, \sigma)$ is a Gaussian with mean μ and variance σ. Analogously, we define the SL0 similarity as [1]

$$s(\mathbf{x}, \mathbf{y}) = D - d(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{D} \mathcal{G}(x_k - y_k | 0, \sigma). \tag{5.2}$$

Using this notion of similarity, we proceed in three steps to learn the user intent and find similar placements, as shown in Figure 5.2:

1. Compute a feature vector for the bounding box B_i of each example placement.
2. Create a statistical model in feature space using linear regression with the SL0 distance on the example feature vectors.
3. Find new candidate placements and compute their similarity by evaluating the statistical model at their feature vectors.

5.3 Geometric Relationship Features

The geometric relationship of a bounding box to the surrounding geometry is described by a feature vector. For this purpose, the bounding box is sampled at a fixed set of points (five points in our implementation, the corners and the center of the bounding box), and the geometric relationship of each point to its surrounding geometry is computed as described in Chapter 4. The feature vector of the bounding box is then the concatenation of the sample feature vectors. In the following, we describe how to compute the feature vector for a single point, and how feature vectors can be compared.

Scene Elements

Recall that the geometric relationships presented in Chapter 4 describe the relationship of a point to *elements* of the scene. Similarly, we define elements to be the scene polygons themselves and elements of their boundary. The polygon boundary is subdivided into "smooth" segments that are comprised of edges without sharp angles and sharp corners that connect the smooth segments, giving us the elements *polygon*, *segment*, and *corner*:

- a **polygon** A, defined by a sequence of vertices $v_k, k \in \{1, \ldots, n_A\}$,
- a polygon **segment** $g_j \in A$, defined as a sub-sequence of consecutive smooth vertices $g_j = v_a, \ldots, v_b, a, b \in \{1, \ldots, n_A\}$. A vertex is called smooth if the dot product of its adjacent edges is above a threshold ($\frac{v_{k+1}-v_k}{||v_{k+1}-v_k||} \cdot \frac{v_k-v_{k-1}}{||v_k-v_{k-1}||} > \epsilon$), otherwise it is called sharp.

[1] See Appendix C for a proof that this similarity can be used as a valid kernel for regression.

- a **corner** c_j of a polygon, defined by a vertex v_{k_j} that is shared by two adjacent segments g_{j-1} and g_j. Corners correspond to sharp vertices.

Relationship Functions

The relationship functions described in Chapter 4 express a geometric relationship between a scene element and a point numerically. Unlike the previous definition, we use points instead of poses, discarding the directional information. Thus, our relationship function is a functional on points and elements:

$$R : \mathbf{P} \times \mathbf{E} \rightarrow \mathbb{R}.$$

We use a slightly different set of relationship functions between a point and an element than in Chapter 4, taking into account the fact that we do not require the pose direction.

- The **boundary distance** is defined as the minimum distance between a point and a polygon boundary: $R_{bd}(p, A) = \min_{x \in b(A)} d(p, x)$, where $b(A)$ is the boundary of A and d the euclidean distance.

- The **bounding-box coordinates** are a class of relationship functions that are defined as the x- and y component of the cartesian coordinates of a point relative to a polygon's bounding box with origin at the bounding-box minimum, the bounding-box maximum and the bounding-box center, for a total of six relationship functions.

- The **normalized bounding-box coordinates** are defined similar to the bounding-box coordinates, but normalized to the width and height of the bounding-box. Also, since the normalized coordinates are invariant to scaling the element, we only use the bounding box minimum as origin, for a total of two relationship functions.

- The **segment distance** $R_{gd}(p, g)$ is the same as the boundary distance, but considers only a segment $g \in A$.

- The **segment arc length** is the normalized arc length between the location of minimum distance to a point p and the start of the segment. If $x = \arg\min_{x \in b(g_j)} d(p, x)$, then $R_{gl} = \frac{t_x - t_a}{t_b - t_a}$, where t_x is the arc length at x and t_a, t_b the arc length at the start and the end of the segment, respectively.

- The **corner distance** is the distance between a point and a polygon corner: $R_{cd}(p, c_j) = d(p, v_{k_j})$.

- The **corner ratio** is the angle that the direction from corner to point p makes with the start of the first polygon segment adjacent to the corner, normalized with the opening angle of the corner: $R_{cr}(p, c_j) = \frac{\angle(p - v_{k_j}, v_{k_j - 1} - v_{k_j})}{\angle(v_{k_j + 1} - v_{k_j}, v_{k_j - 1} - v_{k_j})}$, where $\angle(a, b)$ is the angle between vectors: $\angle(a, b) = \arccos(a/||a|| \cdot b/||b||)$.

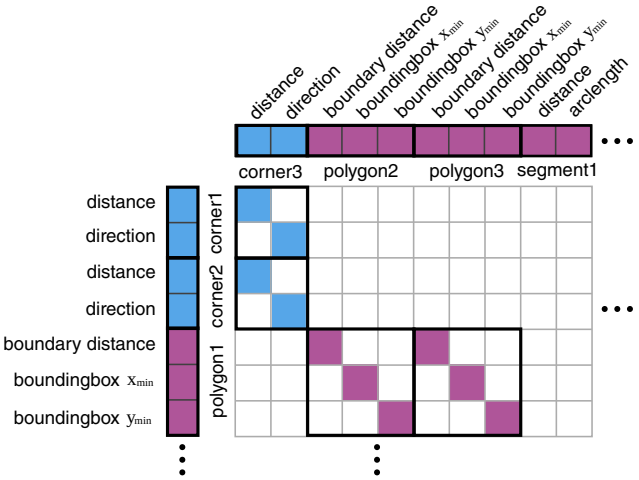


Figure 5.3: Feature vector comparison. We show parts of two feature vectors with features corresponding to corners (blue) and polygons (red). Possible assignments of features are shown in the central matrix. Only features with the same feature type and relation type can be assigned.

Feature Vectors

The set of all relationship function values of a point can be arranged in a vector to give a feature vector. Since the relationship functions are evaluated for each scene element in the neighborhood of a point, the dimension of the feature vector depends on the number of relationship functions, as well as on the number of nearby scene elements. For example, point A may have a distance relationship to two nearby polygons, while point B may only have one nearby polygon.

This leads to the problem that feature vectors with different dimensions are not directly comparable: given two feature vectors $\mathbf{x}$ and $\mathbf{y}$, it is not clear which of the entries of the feature vectors can be compared. Each entry can be described by the pair (relationship type, scene element), e.g., one vector may contain the distance and direction to two corners, while a second vector may contain the distance and direction to three different corners. To compute the SL0 similarity, we need to find a one-to-one assignment between the entries of the vectors. Our strategy is the choose an assignment that maximizes the resulting SL0 similarity under two constraints: First, only entries with the same relationship types and element types can be assigned and second, entries corresponding to a single scene element in vector $\mathbf{x}$ are assigned to only a single scene element in vector $\mathbf{y}$.

The first constraint ensures that only comparable relationship values are assigned. The second constraint effectively reduces the feature-wise assignment to an element-wise assignment. Note that it would be possible to assign the relationships of one element in the first vector to relationships of different elements in the second vector, but we found that this usually only increases clutter without adding interesting placements.

To enforce the first constraint, we partition the feature vectors by element type (like corner, edge, etc.), and consider assignments only between elements of similar type. The second constraint allows us to consider assignments on a per-element level instead of a per-feature-value

level. Note that the feature values between elements of similar type are trivially comparable since they have the same features.

More formally, let $\mathbf{x}_E$ be the subvector of a feature vector $\mathbf{x}$ with features corresponding to a scene element E. Given two feature vectors $\mathbf{x}$ and $\mathbf{y}$ (where we assume a single element type without loss of generality), let $\mathbf{E_x}$ and $\mathbf{E_y}$ be the set of corresponding scene elements. We also define a weight function as

$$S_{\mathbf{x},\mathbf{y}}(E_i, E_j) = s(\mathbf{x}_{E_i}, \mathbf{y}_{E_j}), \tag{5.3}$$

where s is just the SL0 similarity defined in Equation 5.2 with D being the number of relationship functions of the currently considered element type. To solve an assignment problem, the sets need to be of equal size, therefore we fill the smaller set with nil-elements. The weight function is set to 0 if a nil-element is involved. An assignment is then a bijective function $f : \mathbf{E_x} \to \mathbf{E_y}$. We look for the assignment $\hat{f}$ that maximizes the utility function

$$u_f(\mathbf{x}, \mathbf{y}) = \sum_{E \in \mathbf{E_x}} S_{\mathbf{x},\mathbf{y}}(E, f(E)), \tag{5.4}$$

so that $\hat{f} = \arg\max_f u_f(\mathbf{x}, \mathbf{y})$. Note that since each term in the utility function is an SL0 similarity, the utility function is again an SL0 similarity, with D corresponding to the number of features for the current element type times the number of elements in the larger of the two feature vectors $\mathbf{x}$ and $\mathbf{y}$. Maximizing the utility therefore corresponds to finding the assignment with the maximum SL0 similarity. This assignment problem is well studied in graph theory, and we solve it using the *Hungarian algorithm* [56]. Finally, the overall *adapted SL0 similarity* is just the sum of the individual maximum utilities for each element type e, and calculated as:

$$\hat{s}(\mathbf{x}, \mathbf{y}) = \sum_e u_{\hat{f}_e}(\mathbf{x}_e, \mathbf{y}_e) \tag{5.5}$$

5.4 SL0 Regression

The adapted SL0 similarity allows us to calculate the similarity of two placements. Using this, we want to evaluate how well an arbitrary new placement corresponds to the given placements. Since we characterize placements by their feature vectors, we need to operate in feature space, i.e., the space of feature vectors.

The input is the set of feature vectors $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$ corresponding to the given placements, and a feature vector $\mathbf{x}$ corresponding to a new placement. While we could create a measure based on the similarities of $\mathbf{x}$ to the elements of $\mathbf{Q}$ (for example using kernel density estimation), this would overemphasize placements that are given more often by the user. Therefore, we assign a target value t to each $\mathbf{q} \in \mathbf{Q}$, and use these to predict the target value of $\mathbf{x}$. In other words, we have to learn a function $t(\mathbf{x})$ given $\mathbf{Q}$ and $\mathbf{t} = (t_1, \ldots, t_n)^{\mathrm{T}}$. Target values can be between 0 and 1. Thus, the user can specify the preference for a placement. If no target values are given, we set all $t_i = 1$.

In machine learning, kernel regression is a method for learning functions that has several advantageous properties for our setting. First, it uses no fixed set of basis functions that need

to completely cover the space of possible functions – instead, it adapts to the given data points. Second, it can be expressed using a kernel function quantifying the similarity between two input elements. The kernel function can be chosen freely under some validity constraints. The disadvantage of kernel methods is that the evaluation can be slow because all input elements are used to represent the target function. However, in our case, the number of placements is low, so this is not a concern.

More concretely, we follow the definition of Bishop [10] of kernel regression, where the target function is calculated as

$$t(\mathbf{x}) = \mathbf{k}(\mathbf{x})(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{t}. \tag{5.6}$$

Here, λ is a regularization factor to avoid overfitting, and $k(\mathbf{x}, \mathbf{y})$ is the kernel function, which is used to define the terms $\mathbf{k}(\mathbf{x}) = (k(\mathbf{q}_1, \mathbf{x}), \ldots, k(\mathbf{q}_n, \mathbf{x}))$ and $\mathbf{K}$, the gram matrix of the kernel with:

$$K_{ij} = k(\mathbf{q_i}, \mathbf{q_j}). \tag{5.7}$$

For the kernel, we use the adapted SL0 similarity defined in Equation (5.5):

$$k(\mathbf{x}, \mathbf{y}) = \hat{s}(\mathbf{x}, \mathbf{y}) \tag{5.8}$$

with a kernel size proportional to the size of the placement. We will discuss kernel size in more detail in Section 5.7. We prove in Appendix C that this is a valid kernel function.

5.5 Finding Candidate Placements

Given the learned model $t(\mathbf{x})$, it is possible to determine the quality of any given placement with feature vector $\mathbf{x}$. However, the space of all possible placements in the scene is four dimensional, and therefore an exhaustive search is not possible. For this reason, we need to reduce the set of candidate placements before evaluating the learned model.

Recall that the feature vector of a placement is a combination of the feature vectors of a set of sample points on the bounding box of the placement. If we start by looking for good placements of an individual sample on the bounding box (e.g., the lower-left corner), we can make use of the fact that the search space for each separate sample point is only two-dimensional. Let $\mathbf{p}_{ij}$ be sample point i of example placement j. We learn a statistical model $t_i(\mathbf{x})$ for each set of corresponding sample points $\mathbf{P}_i = \{\mathbf{p}_{ij} | j = 1 \ldots m\}$, where m is the number of example placements. For this, we use the feature subvectors $\mathbf{x}_{ij}$ corresponding to sample point $\mathbf{p}_{ij}$ and use the same methodology as in Section 5.4.

Since the sample point positions are two dimensional, it is possible to compute the predicted quality for a dense grid of positions in the scene. We find the best positions $\hat{\mathbf{P}}_i$ for each statistical model by non-maximum suppression with a small fixed radius on the grid of target values [51], and discard maxima below 10% of the largest maximum.

A full placement (i.e., bounding box) has four degrees of freedom, and thus it is possible to reconstruct placements by combining pairs of positions from different $\hat{\mathbf{P}}_i$. For each pair of positions $(\hat{\mathbf{p}}_i, \hat{\mathbf{p}}_k)$ with $\hat{\mathbf{p}}_i \in \hat{\mathbf{P}}_i$, $\hat{\mathbf{p}}_k \in \hat{\mathbf{P}}_k$ and $i \neq k$, we construct one candidate placement,

Figure 5.4: Candidate placements and ranking. Given the facades and the red example balcony placements shown in (a), candidate placements are found using individual sample points (b). The score for the lower left corner is shown as heat map. Ranking the candidates based on individual sample scores results in undesirable placements (c, marked by red arrows, highest score is red, lowest is blue). The combined model includes information about the entire placement and the combined score gives better results (d). Finally a user can adjust a score threshold to get only the placements most similar to the examples (e).

except for the degenerate cases where the two points lie on the same bound-box edge. See Figure 5.4 for an example of candidate placements on a simple facade.

For each candidate placement with full feature vector $\mathbf{x}$, we then calculate its target value $t(\mathbf{x})$ (which uses all sample points on the bounding box), thus obtaining a list of ranked placements. Using the full feature vectors, only placement that are consistent across samples get a high score. Figure 5.4 shows the difference between a ranking based on the feature sub-vectors (c) and the full feature vectors (d). In this example, using the full feature vector ensures that the lower left and upper left corners of a placement have relations to the same window, thereby avoiding bad balcony placements. The list of candidate placements can still be quite exhaustive and contain low-quality placements. Furthermore, the target value only takes into account the geometric relations to surrounding objects, but not geometric properties of the placements itself. Therefore, we apply a number of filters and modifiers to it:

- Remove placements with a target value lower than 10% of maximum score, to filter out low-quality placements.
- Penalize placements where the bounding box gets deformed too much, described in more detail below.
- Remove placements with negative width/height to avoid mirrored placements (if desired by the user).
- Remove overlapping placements, starting with the lowest score.
- Further application-specific filters, which will be described later, such as removing placements that intersect geometry with labels that were not intersected by the example placements.

To penalize deformations, we compute the change in the width/height ratio and the area. Let w, h, A be width, height and area of a placement, and w_e, h_e, A_e the corresponding average

values over all example bounding boxes $b \in \mathbf{B}$. Then the changes are rated by Gaussians with user-specified tolerance tol_{wh} and tol_A:

$$t'(\mathbf{x}) = t(\mathbf{x})G(\log(\frac{w/h}{w_e/h_e}), 0, tol_{wh}^2)G(\log(A/A_e), 0, tol_A^2). \tag{5.9}$$

The adjusted target value t' is then used to present the user a set of placements with descending placement quality, in a form determined by the application (e.g., color coding). Note that we employ only this very basic form of shape matching since the focus of our method lies on matching the relations of placements to other scene elements, rather than the geometric properties of the placements themselves.

5.6 Applications

The proposed algorithm can handle *2D applications* like building placements in the parcels of an urban street plan without modifications. Here, the parcels are given as polygons and building footprints can be modeled by either single polygons or the union of several polygons. After giving several example placements, each building polygon can be transferred to all remaining parcels. Even in 2D placement tasks, some extensions like labels or hierarchical subdivision are useful and described in the following.

Most applications, however, involve *3D geometry*, where placements are done on surfaces. Furthermore, if the placed objects are assigned a height, mass modeling becomes possible. For 3D scenes, we thus introduce several extensions, in particular surfaces to act as domains for placement, and extrusions and Boolean operations to allow for mass modeling.

Finally, we show how to record learned results to transfer them to other scenes.

These extensions allow us to apply the method to implement several steps in a 3D city modeling pipeline, including building placement on parcels of a street network, mass modeling from the building footprints, facade modeling, and furniture placement. Results for this application will be shown in Section 5.7.

2D Extensions

The following two extensions are useful in pure 2D placement tasks as well as in the general 3D case described further below.

Element Arrays Instead of scaling elements to fit their placements, we can also keep their original size and repeat them instead, resulting in an array of elements for placements of larger size. By specifying a minimum distance between the repeated elements, we can create large arrays of objects like windows in a single operation. The choice between scaling or repeating can be specified separately for the width and height dimensions.

Labels Since our focus does not lie in object recognition, we assume that all elements of the scene are labeled according to their semantics (e.g., window, door, facade), and set entries in the weight function $S_{\mathbf{x},\mathbf{y}}$ (see Equation 5.3) for pairs of elements with different labels to zero. This

avoids cases where two placements might be considered highly similar because each of them has a similar geometric relation to a feature, but the two features are of different object types (e.g., a window and a door).

Polygon Hierarchy Usually, complex layouts such as facades or street plans can be naturally divided into several disjoint areas where the elements are placed independently. For example, using a structure where each parcel in a street layout resides in a separate node, the learned layout is not influenced by neighboring parcels, which is usually desirable. To generalize this notion to arbitrary poygonal scenes, we create a hierarchy of polygons, as suggested by Guerrero et al. [42]. A polygon A_p is parent of a polygon A_c if it contains A_c or if the polygons intersect and A_p has a larger area. The parent polygon of a point q is defined as the smallest polygon containing the point. Additionally, each polygon has a main orientation. All elements and placements share the orientation of their their parent. When computing the feature vector $\mathbf{x_q}$, only relationships to elements that have the same parent as q are considered.

3D Extensions

Since our method operates on two-dimensional domains, we cannot apply it directly to three-dimensional city models. Instead, we apply our method to planar parametrized surfaces in the scene and introduce operations to build interesting 3D models from planar polygons.

Surfaces are either created and parametrized automatically as part of the mass modeling operations described below, or the surfaces and their parameterization are defined by the user. All elements in the scene are placed on a surface, e.g., a house is placed on the parcel surface, and a window is placed on the facade surface. On such a surface, the elements are represented as two-dimensional polygons, corresponding to the projection of the elements onto the surface. Relations are only computed inside a surface, not between surfaces.

Extrusions New 3D elements may also be introduced through the extrusion of polygons. For example, the layout of a building may be extruded to create the basis for a building model. Planar surfaces are automatically found and parametrized on the extruded mesh. Polygons are always extruded in a direction normal to their surface with a height specified interactively by the user. An optional random multiplier can be applied to the height to add variation to the resulting elements. Straight boundary segments of the polygon correspond to planar surfaces on the extruded 3D element. Finally, a roof or a flat cap can be added to the bottom or top side for a more realistic appearance.

Boolean Operations To construct more interesting 3D shapes, we can combine several elements using boolean operations such as unions or subtractions. This allows the creation of more complex building mass models with variation introduced by the placement of the individual elements. Boolean operations are applied in two ways: Either by merging or subtracting one or several elements with/from their surface, or by or merging intersecting elements on the same surface.

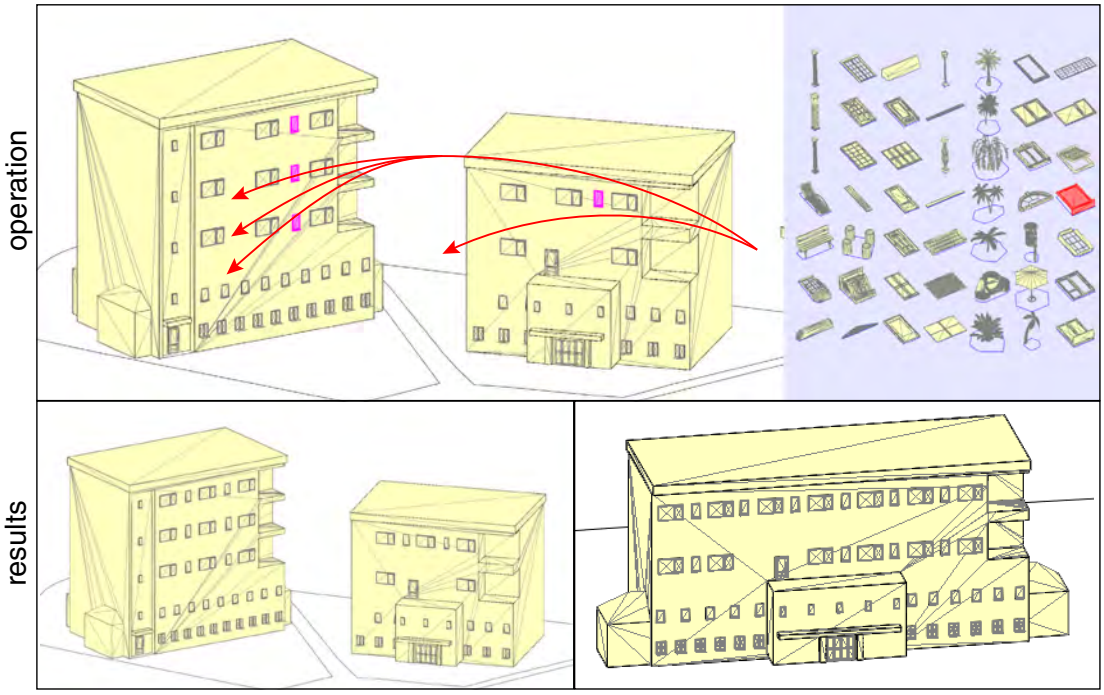


Figure 5.5: Typical operations performed in our editor. In a placement operation (top row), an element is dragged from the element library (light blue) to a surface in the scene and resized if necessary. The same operation is repeated for each example element. Results of the propagation to several buildings are shown in the bottom row.

Operations Tree

Each modeling operation is recorded in an *operations tree*. The result of a modeling session can be applied to a different scene without user interaction, for example, applying a specific style to a given basic building shape (an example might be a saloon or a run-down hotel) or creating a small diverse city section from a given set of parcels. The specific operations are:

Placement, which finds new placements in a given input scene from the bounding boxes of a set of example elements. The recorded information consists of the mesh that is to be placed, as well as the learned target functions $t(\mathbf{x})$ and $t_i(\mathbf{x})$ (i.e., the example placements are discarded as their information is now stored in the learned model, and can thus be applied to any input scene).

Extrusion, which extrudes a given set of polygons to create new meshes and surfaces.

Boolean Operation, which merges or subtracts an element from its own surface, or subtracts/merges all elements on the same surface that intersect the given element.

Selection, which selects elements either by label, or orders them randomly into n disjoint sets.

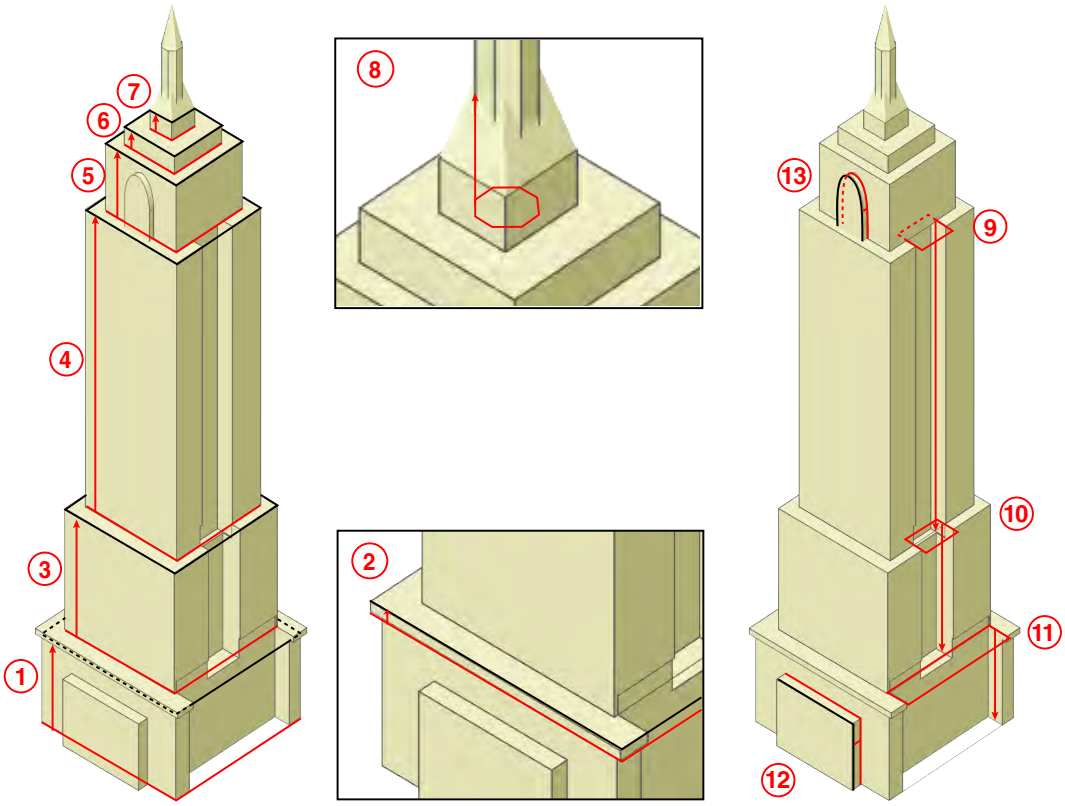


Figure 5.6: Mass model editing of a skyscraper. The mass model shown above is created with several edit operations in the order indicated by the red circles. In each operation, the red shape is placed and extruded along the red arrow. In steps 1-8, 12 and 13, this is followed by a Boolean union with the parent surface. In steps 9-11, a Boolean subtraction is used instead. Step 7 and 8 cap the extruded mesh with a roof.

The tree is defined by connecting the input elements and output elements of corresponding operations. The tree limits the scope of operations to a specific branch of the tree, so that by using the Selection operation with disjoint sets and applying a different operation to each set, different branches of the operations tree can be applied to different sets of elements, increasing the variation of the resulting scene.

Workflow

We have created an application prototype that implements the described operations. Since 2D scenes are a special case of a 3D scene with just one surface, this application can handle a number of different scenarios, in particular all the mentioned steps of a city modeling pipeline: building placement in parcels, mass modeling of buildings, facade generation, and interior modeling.

In a typical workflow, creating a scene starts with a basic layout, for example, a layout of the parcels in a city scene or the layout of rooms for interior modeling. In a complex scene, the user typically only works on a small representative subset of this basic layout. Geometry is

created by dragging labeled meshes or polygons from an object library to surfaces in the scene, resizing them, and then propagating them to the rest of the scene using the placement operation. Figure 5.5 shows a typical operation in our prototypical editor. Propagated polygons can then be extruded to 3D shapes and more complex shapes can be modeled using Boolean operations. In Figure 5.6, we show the operations needed to create the mass of the skyscraper. In each operation, the red shape is placed and extruded in direction of the red arrow, followed by a Boolean union or subtraction from its containing surface. Most of the extrusions have a randomized height, and steps 3,5 and 10 are optional branches of the operations tree, adding variation to the resulting models. Finally, operations 3-6 are performed with different shapes, like octagons, in different branches of the operations tree.

Before performing an operation, the user selects a leaf node in the operations tree. This leaf node provides the input of the new operation and becomes its parent. The finished operations tree can then be applied to the large scene without user interaction. If needed, the operations can be split into several subtrees that can be created independently as long as the operations do not influence each other. For example, it is possible to work on the base and the top of a skyscraper separately and then merge the separate trees before applying them to a new scene.

5.7 Results

Example Scenes We demonstrate our method on three example scenes modelend in a simple Matlab prototype of our method, in each of which various steps of a city-modeling pipeline are implemented using our method:

A *large city scene* shown in Figures 5.7 and 5.8, where we populate the parcels in a street layout of Tempe, AZ, with buildings of three different types. All parcels have an orientation that points towards the street-facing side and are labeled either 'house' or 'apartment building', corresponding to the type of building placed into the parcel. Three types of houses are generated in the parcels: two types of family houses and one apartment building type. Three operations trees, one for each type, were generated in three editing sessions to implement the mass-modeling and facade generation steps. The Example scenes contained 7 houses each and 2–6 examples were provided for each operation. Note how the building shape and consequently the facade details are determined by the parcel size, creating varied house models. Other sources of variation are the random selection of alternative branches in the operations tree and the use of random extrusion heights, as described in Section 5.6. A total of 135 operations (including selections) were used to create all three building types.

A scene with *New York style skyscrapers*, where we apply an operations tree to rough footprints of the skyscrapers, is shown in Figure 5.1. The operations tree was generated in four editing sessions, one for the building mass and one each for the base, mid and top facades. Nine sky scrapers were part of the example scene and 2–8 examples were provided in each operation. Variation is generated in part through the skyscraper footprints and in part through random selection operations and extrusions with random height. A total of 146 operations were used in the four editing sessions.

A *kitchen scene*, where we apply an operations tree to single rooms. See Figure 5.9 for 16 kitchen scenes, each viewed from two different angles. Typically, in interior modeling, a given

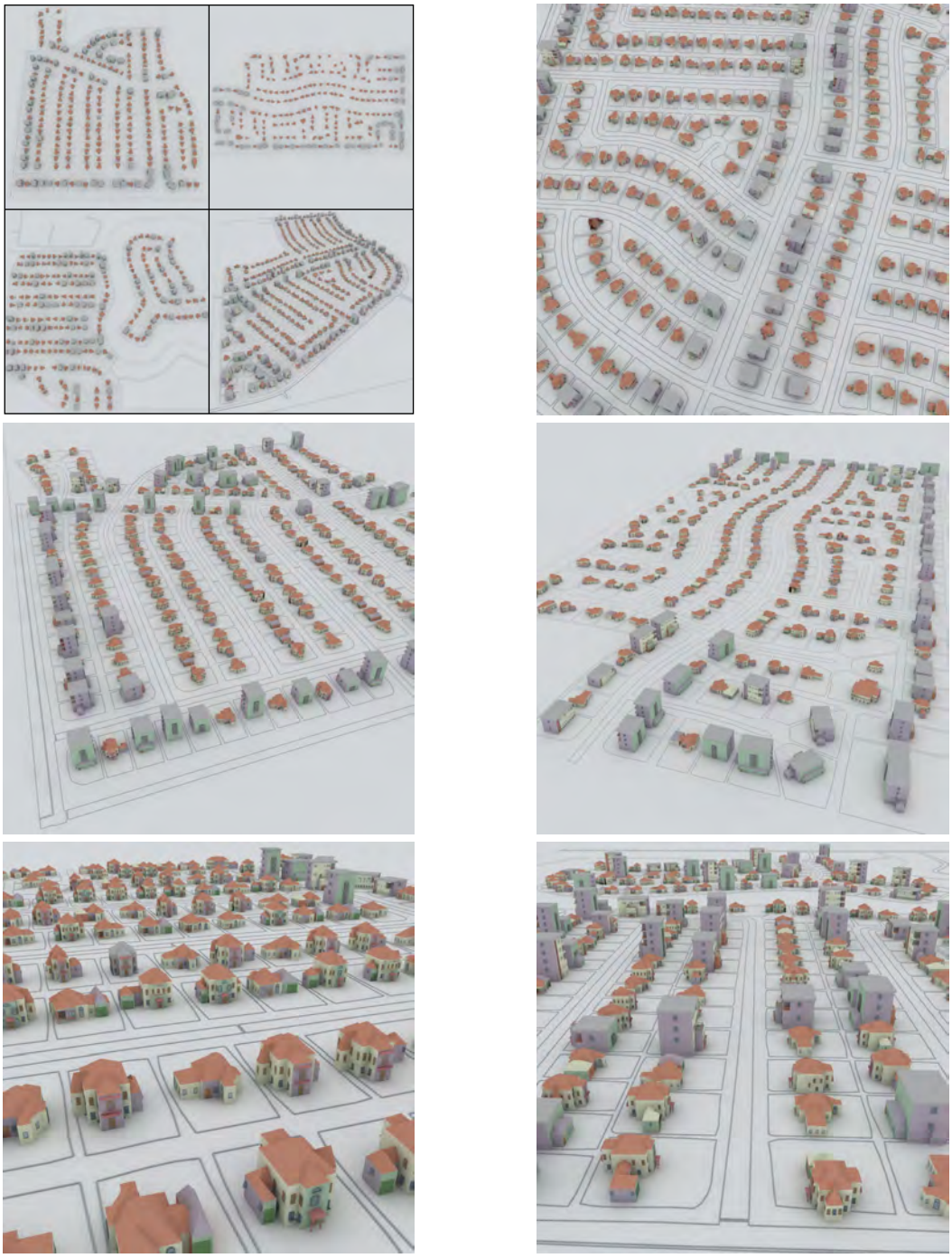


Figure 5.7: City scene. Three different types of houses (condominium building, two-story house and one-story house) were generated in three edition sessions. The resulting operations tree was applied to the four city maps shown as lines on the ground of the overview shots on the upper left. Given a set of labeled parcels, large city areas can be generated by our method without further user interaction.

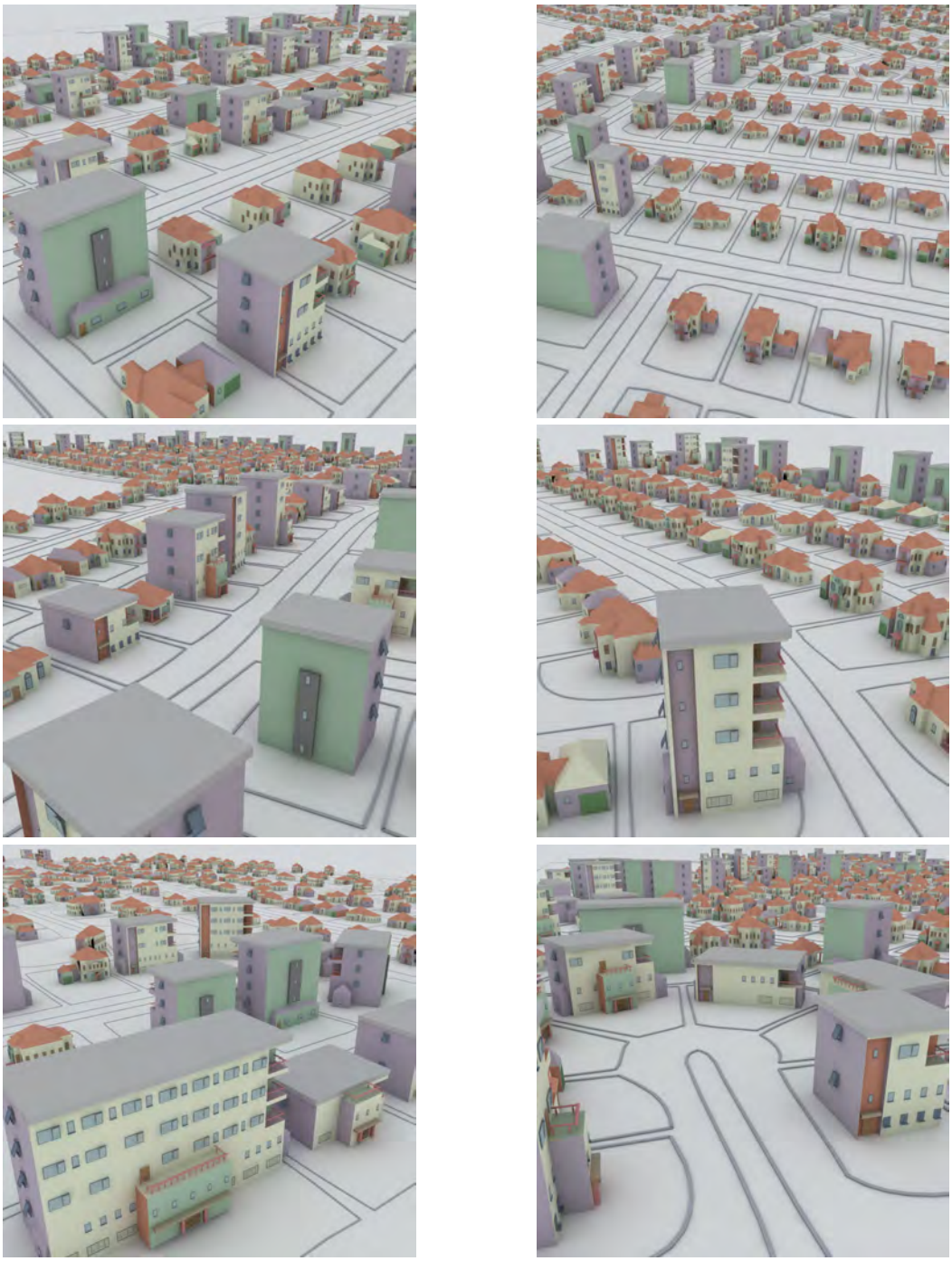


Figure 5.8: Additional views of the city scene. Close-ups of individual streets and houses are shown.

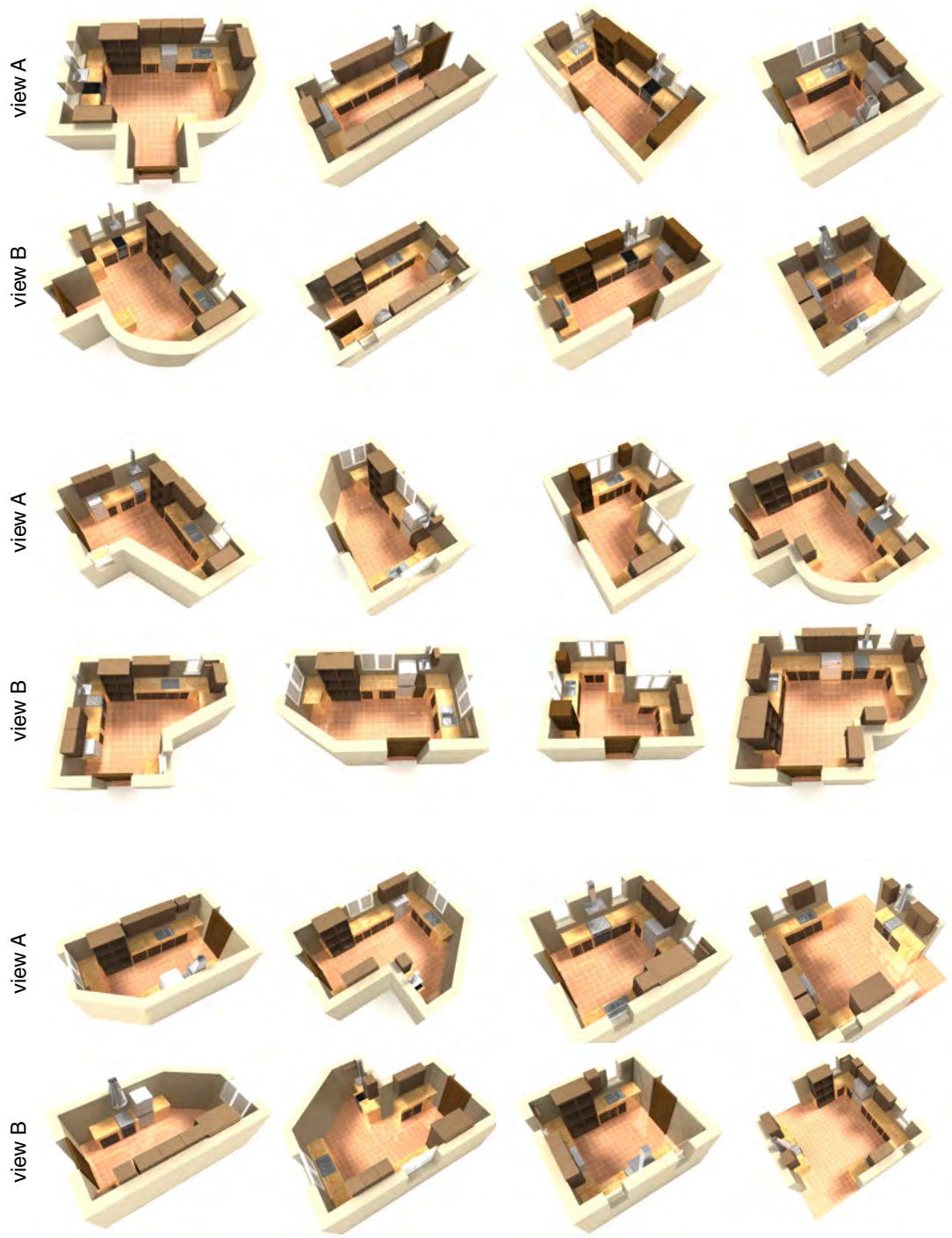


Figure 5.9: Kitchen interiors generated by our method. Starting from a labeled room shape, we generate the 3D model of the kitchen, including furniture, walls and windows. Two views are shown for each kitchen.

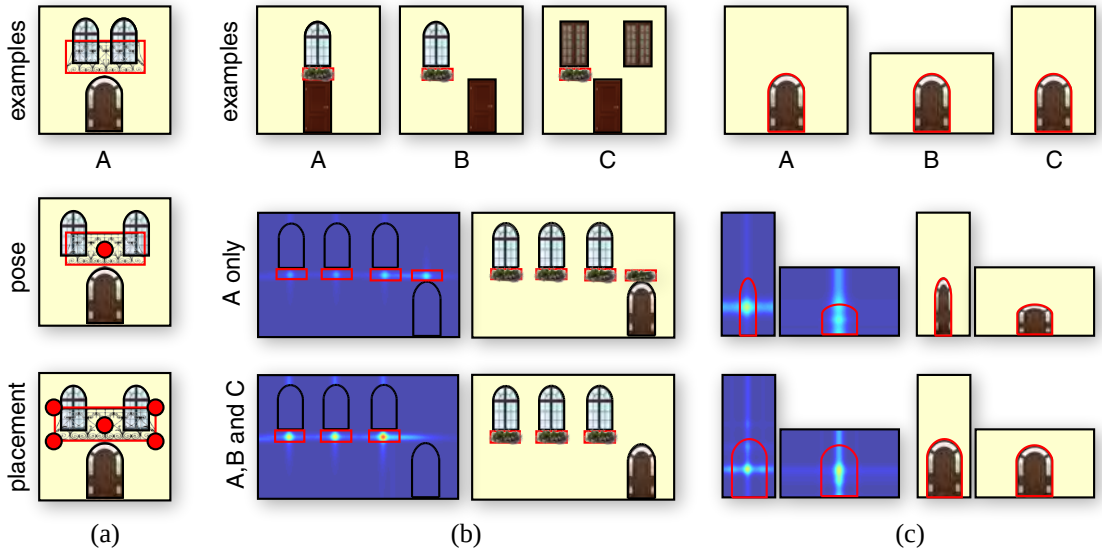


Figure 5.10: Comparison of poses versus placements, as well as single examples versus multiple examples. In (a) we show that only using a singe sample point per element (center row, red dot) does not allow for adaption of the shape to the surrounding geometry. With multiple sample points adaption is possible (bottom row). Providing only a single example in (b) results in ambigous placments for the flower boxes (center row), either above a door or below a window. Using additonal examples resolves the ambguity (bottom row). In (c) doors are propagated to empty facades of different sizes. With only a single example it is unclear if the doors should mantain their size or adapt to the facade. Providing additional examples clarifies the user intent.

empty room is filled with furniture, so the starting point of this example is a polygon defining the shape of the room and a label indicating the walls that are to contain a door. The example scene contained 9 kitchens and 2–6 examples were given in each operation. Note how the furniture adapts to the shape of the room creating variation. Additionally, alternative branches in the tree were used to place different window styles and furniture models. The operations tree consists of 85 operations.

Computational Complexity The computational complexity of the method mainly depends on the number of example placements N_e, the number of points N_p in the position grid defined in Section 5.5, and the maximum number of elements N_f with the same label and type in the example feature vectors. Since we use a kernel method, the feature vectors of all grid points have the be compared to the feature vectors of all example placements. For each pair of feature vectors, one assignment problem has to be solved for relevant subvector as defined in Equation 5.3, with a maximum size of a subvector given by the number of elements with same type and label. The resulting computational complexity is $O(N_e * N_p * N_f^3)$. In practice, N_f does not depend on the scene size, since increasing the scene complexity usually only increases the number of independent surfaces or hierarchy polygons, while the number of mutually dependent elements inside these areas stays relatively constant. For example, when adding more houses to a city

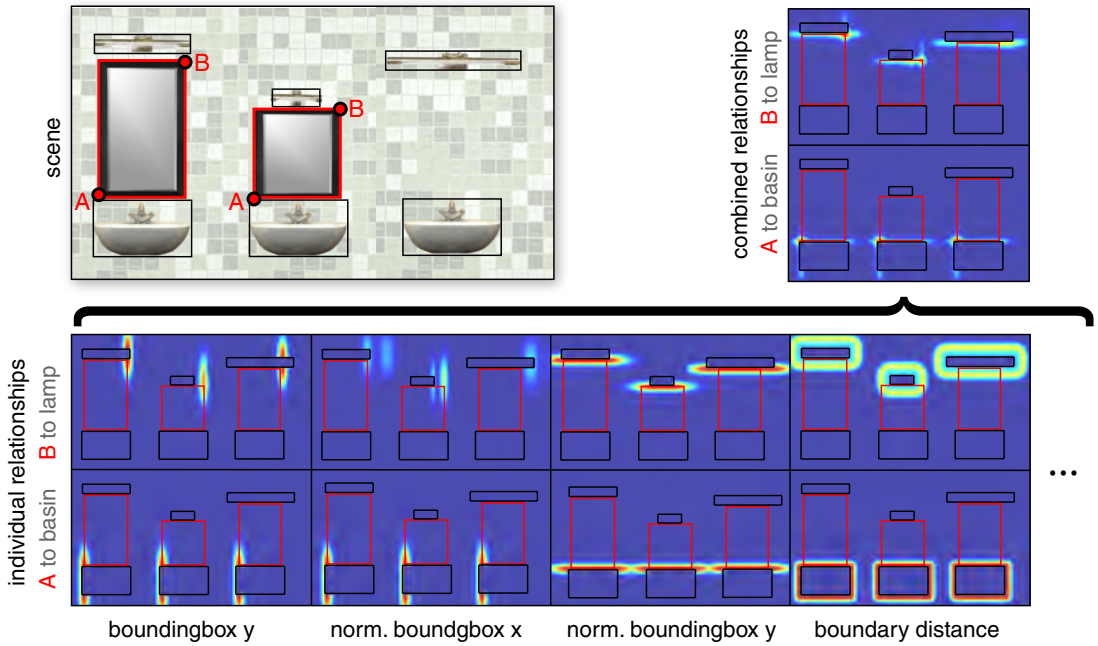


Figure 5.11: Learned importance of indivdual relationships. A mirror placement between a basin of fixed size and a lamp of varying width is learned from two examples. Scores of individual relationships are shown as heat map on the right. For more information on the relationship types, see Section 5.3.

scene, the number of facades increases, while the number of elements inside each facade remains relatively constant. N_e is usually independent of the scene size as well, typically a user provides 2-10 example placements. This leaves only the term N_p that directly depends on the scene size, giving our method a linear time complexity in practice.

Comparison to Previous Work Our method is based on the geometric relationship functions described in Chapter 4. In the following we discuss the main differences to this method. First, we propagate placements based on relationships at multiple points instead of poses based on relationships at a single center point. In our scenes it is often necessary to adapt the size of an element to the surrounding geometry, for example to adapt the width of a balcony to the width of the facade, as shown in Figure 5.10a. Pose propagation is rigid, and the relationships at a single center point do not provide enough information to adapt height and width of an element. Our method computes relationships at several points on a shape and adapts height and width to maximize the relationship similarity at each point. Second, we learn from multiple examples instead of a single one. This allows us to disambiguate placements, i.e., to learn which relationships are important. Consider Figure 5.10b, here a flower box is placed under a window that happens to be above a door. With only a single example, the relation to the door and to the window are equally important, and consequently flower boxes are also placed above all doors. With multiple examples, the user intent becomes clear and flower boxes are only placed under windows.

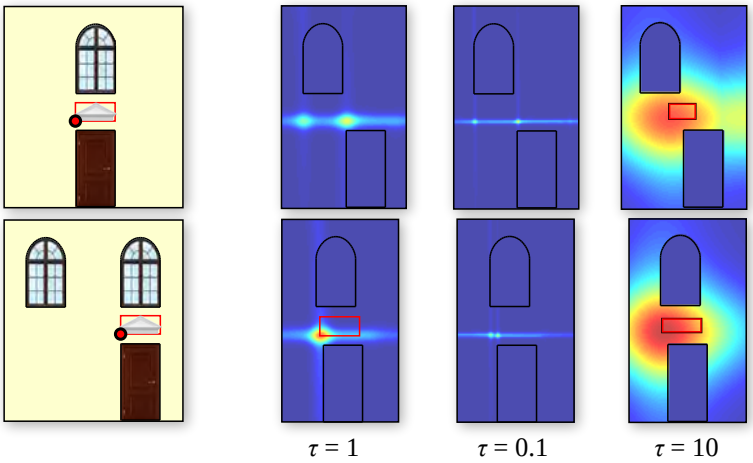


Figure 5.12: Effects of changing the kernel size. In two examples, a small roof is placed between a door and a window (left). The score for the lower left edge point is shown as heat map. The upper row window is strongly misaligned, the lower row window only slighty. τ denotes the kernel size. With a small kernel size, both misalignments are too strong to be considered similar to the examples, while with a big kernel size, both misalignments are considered similar.

Relationship Influence The influence of each relationship type on the score of a point is determined by the kernel. In Figure 5.11, we show the most important relationships learned from two sample points of two example placements. Note that the lamp varies in size, but like the example placements, the mirror remains at the size of the basin. The effect of learning the examples is observable in the low weight of the normalized bounding box width relationship to the lamp (second column, top row), compared to the same relationship in the bottom row.

Kernel Size The size of the SL0 Kernel is determined by the variance of the Gaussians in the SL0 similarity. Each relationship type defines a variance based on its range of values, e.g., a directional relationship needs a different variance than a relationship that computes an absolute distance. Relationships based on an absolute distance use the bounding-box diagonal of the placement to normalize the variance. We can additionally multiply the variances with a constant factor τ to control the precision of the placements. Figure 5.12 shows the effect of changing τ. Lower values increase precision and decrease tolerance, while higher values do the opposite. A good value of τ is a consideration between precision and tolerance. We use $\tau = 1$ in all our examples.

Limitations

When creating an operations tree and learning example placements, the example scene should be representative of the types of scenes the operations tree is intended for. Consider Figure 5.13, where an operations tree was generated from single-floor buildings only. When applying the operations tree to buildings with two floors, there is ambiguity that is not resolved by the examples. For example, should the entrance be placed in the first or second floor, or should it cover both

Figure 5.13: Results of applying an operations tree learned from unrepresentative examples. The operations tree for a family house was learned on single-floor buildings (bottom row). When applied to two-floor buildings, there is ambiguity that results in unexpected element placements (top row). Providing addtional examples on two-floor buildings would help resolve the ambiguity.

floors? Since our method operates on a purely geometric level, these questions can only be resolved by providing additional examples.

Since our method operates on bounding boxes only, the orientation of elements is not considered. It is not possible to learn a placement that aligns the orientation of an element to some neighboring element. However, in city scenes, elements that share the same polygon hierarchy parent often share the same orientation. For example, elements on a facade are all aligned with a common orientation. In future work, we would like to generalize our placements to more general oriented bounding boxes, although the additional degrees of freedom would make the search for candidate placements more difficult.

5.8 Summary

In this chapter, we have presented a method for placing shapes by example in 2D scenes. In contrast to the method presented in the previous chapter, which is limited to a single example and a single representative point, we can take multiple example placements into account, as well as the rough shape of the examples. This is made possible by using kernel regression to learn the user intent, and by introducing a new metric on feature vectors that can deal with feature vectors of different types and dimensions. We further generalize the 2D placement operation to 3D scenes using extrusion and Boolean operations, which makes it possible to apply the method for a wide range of modeling tasks. Using an operations tree that can record the operations carried out by the user, modeling sessions can be applied to different scenes automatically. For example, we have demonstrated how several steps in a city modeling pipeline can be efficiently carried out using our application prototype.

Different to previous work in modeling-by-example, we use individual operations as examples instead of complete scenes. This gives the user more direct control over the result, and the information gained by having knowledge of individual modeling steps allows a better prediction of the user intent. Additionally, individual operations have fewer degrees of freedom than

complete scenes, making it possible to successfully learn from fewer examples. Consequently, our method does not require a large database of example scenes.

CHAPTER 6

Summary and Conclusions

Shape modeling is a broad area of research where a large variety of approaches can be successfully applied to make a modeling task more efficient. We have discussed current work in two main research directions: analyze-and-edit methods and modeling by example, as well as edit propagation as a specialization of modeling by example. Procedural modeling is a third major area of research, but is less related to our work. Although it is difficult to formulate a concrete goal for shape modeling, one desirable long-term result is a method that has a similar level of autonomy as a human modeler, where the user only needs to provide high-level directives or examples, as if instructing a person. This touches on many different areas in computer science, including machine learning, artificial intelligence, shape understanding and shape similarities. Specifically, learning a high-level specification of the modeling task and domain-specific knowledge from as few examples as possible is still a challenging task.

Although the presented work provides new ideas to tackle this challenge for scenes containing deformable objects or shape arrangements, allowing for novel modeling operations that can significantly reduce the effort to model these scenes, the wide variety of semantically similar but geometrically distinct objects, or the more complex relations found in many scenes that may not only depend on geometric information still pose major problems that require finding additional solutions.

6.1 Key Contributions

The central idea of our work is to focus on geometric analogies, that is, similarities between shape relations, as opposed to similarities between shape properties. We propose several novel geometric analogies that we use as main component in edit propagation methods, allowing for modeling operations that are not achievable with the current state of the art and that can help to significantly reduce the modeling effort for scenes containing deformable objects or shape arrangements. Specifically, we provide key contributions in two main areas:

Edit Propagation in Deformable 2D Shapes Edit propagation in deformable shapes is a difficult problem, because it typically requires establishing a dense mapping between the shapes. Matching deformable shapes is known to be hard for several reasons: adequately modeling the deformations and finding matched regions that are related by deformations with many degrees of freedom while robustly handling noise and missing model parts are challenging problems. In our work, we present a novel similarity measure called *Transformation Parameter Similarity* that can be used to efficiently match regions related by strong deformations that are locally approximately structure-preserving, such as strong bending or tapering. We show that unlike previous approaches, this method is robust to occlusions and can handle sparse and uneven point correspondences in strongly deformed shapes. A dense mapping can be established between similar shapes, which allows propagating local edits or properties such as texture.

Edit Propagation in 2D Shape Arrangements While establishing a dense mapping between shapes is one possible method to propagate edit operations, it is not always applicable. When editing shape arrangements, similar locations in these arrangements can be identified even if the arrangements themselves are dissimilar. This is evident in floor plans, where rooms may have different shapes, but similar furniture placements can be found in each room nonetheless. Shape matching methods cannot be used to find these locations. We propose a method based on simple *geometric relationships*, such as the distance to a shape boundary or the direction to a shape corner. Similar locations are defined as points that have as many of these simple geometric relationships in common as possible. This conceptually simple method allows us to efficiently find similar locations in dissimilar shape arrangements. We show that this simple geometric analogy can be used in an edit-propagation framework to populate large scenes like floor plans with plausible furniture layouts consisting of hundreds of objects using relatively few edit operations.

In further work, we show that using multiple example placements for edit propagation instead of a single placement can significantly improve prediction of the user intent. Additionally, we allow the placement of bounding boxes, instead of simple points with an orientation. Here multiple example placements, which define position, width and height of the bounding box, are given, and a probabilistic model is trained on simple geometric relationships of the box to the shape arrangements. This allows placing objects that adapt their width and height to best fit the given scene. Finally, we present extensions that enable propagation and generation of shapes in 3D scenes. We show that this method can be used to generate large and complex city scenes and interiors using few edit operations.

6.2 Outlook

An interesting future direction in the area of deformable shape matching is to expand the definition of our Transformation Parameter Similarity to include branching shapes, which are represented by branching surfaces in transformation space. This would allow texturing branching structures like trees for example. Additionally, adapting this method to other types of transformations should be straightforward. Also, the core of our method is input agnostic, suggesting an extension to raster images and 3D shapes as possible areas for future work. Applying Transformation Parameter

Similarity to raster images would only require reasonably stable image feature descriptors, and a 3D extension might even benefit from the richer information provided by the 3D surface.

To model more complex relations in scenes consisting of shape arrangements, it would be interesting to investigate relationships that relate to more than one feature, for example to encode relationships like "between feature A and feature B". Another promising direction could be to examine the relation between edit propagation and modeling by example where entire scenes are given as example. For this purpose, it would be interesting to create a method that propagates the placements of all objects in a room at once and automatically handles mutual constraints to eliminate the dependence of the resulting object arrangements on the order of operations.

A more difficult problem, a solution to which could greatly help simplify modeling tasks, is to propagate placements of objects that are allowed to adapt in more than just their width and height to the surrounding shape arrangement. This could be realized by capturing additional relationships between individual points of the example object. Thus, more complex placement operations, such as placing a longer carpet that could adapt to the shape of the target room, could be accomplished.

6.3 Conclusions

Shape modeling is an active area of research, and there is certainly no lack of problems to solve. Increasingly complex scenes like cities, terrains, or even whole planets need to be modeled. Methods need to simplify these tasks to make them feasible, while still producing the results that the modeler expects. Like in the related area of shape similarities, the goal of this research is not well-defined, but rather depends on the human understanding of shape. A good method is defined as producing intuitive results, that is, predicting the modeler's intentions and expectations, given just a limited amount of geometric data. While this is an ill-defined problem that does not have an optimal solution, several current methods find partial and approximative solutions that are useful to assist the user in modeling tasks. The work presented in this thesis presents conceptually simple geometric analogies in edit propagation frameworks that can be used to significantly simplify the modeling of complex scenes.

APPENDIX A

Local Contour Descriptors to be Used With the TPS Method

As mentioned in Section 3.3, the local descriptors used for our method based on Transformation Parameter Similarity are interchangeable. In this chapter, we describe one possible choice of local descriptors for shapes given as contour segments, based on the well-known Shape Contexts [7].

To define the placement of the descriptors, we sample the shape contours with a discrete set of points. Sampling is performed depending on local contour curvature, creating more points in regions of high curvature [22]. In order to prevent straight lines from being undersampled, a fixed minimum distance (about one percent of the major image dimension) to neighboring points along a contour is always respected. The orientation of each descriptor is aligned with the local tangent at the contour sample. When comparing two descriptors, both possible orientations are tested. To determine the scale of a descriptor at a contour sample, we compute a scale-invariant average of the contour curvature around the sample. To make the average scale-invariant, we cannot use an averaging kernel with a fixed arc length as radius, since the arc length is not scale-invariant. Instead we fix the curvature integral, which is invariant to scale, and measure the arc length of a contour interval with fixed curvature integral. Intuitively, we measure how far an imaginary person would need to walk along to contour, starting at the contour sample, to turn by a fixed given angle (the absolute angle is taken; left and right turns sum up). When scaling a shape, this measure scales proportionally. The walk distance corresponds to the arc length a and the turn angle to the fixed given curvature integral c along the contour. The scale is then proportional to $\frac{a}{c}$. We found that this value is a robust measure of the contour line scale.

Now that the position, scale and angle of all descriptors is defined, we proceed to compute the descriptor signature, i.e. a vector that describes the geometry in the local neighborhood of the descriptor. The Shape Context descriptor [7] provides such a signature and is known to work well on shapes given as point sets. We describe a slightly modified version that is more robust to background clutter and slight variations in the position of contour lines. The local area around a point is discretized into N bins, each bin holding the weighted average of all contour tangents inside a fixed distance from the bin center and a weight describing the distance of the bin center

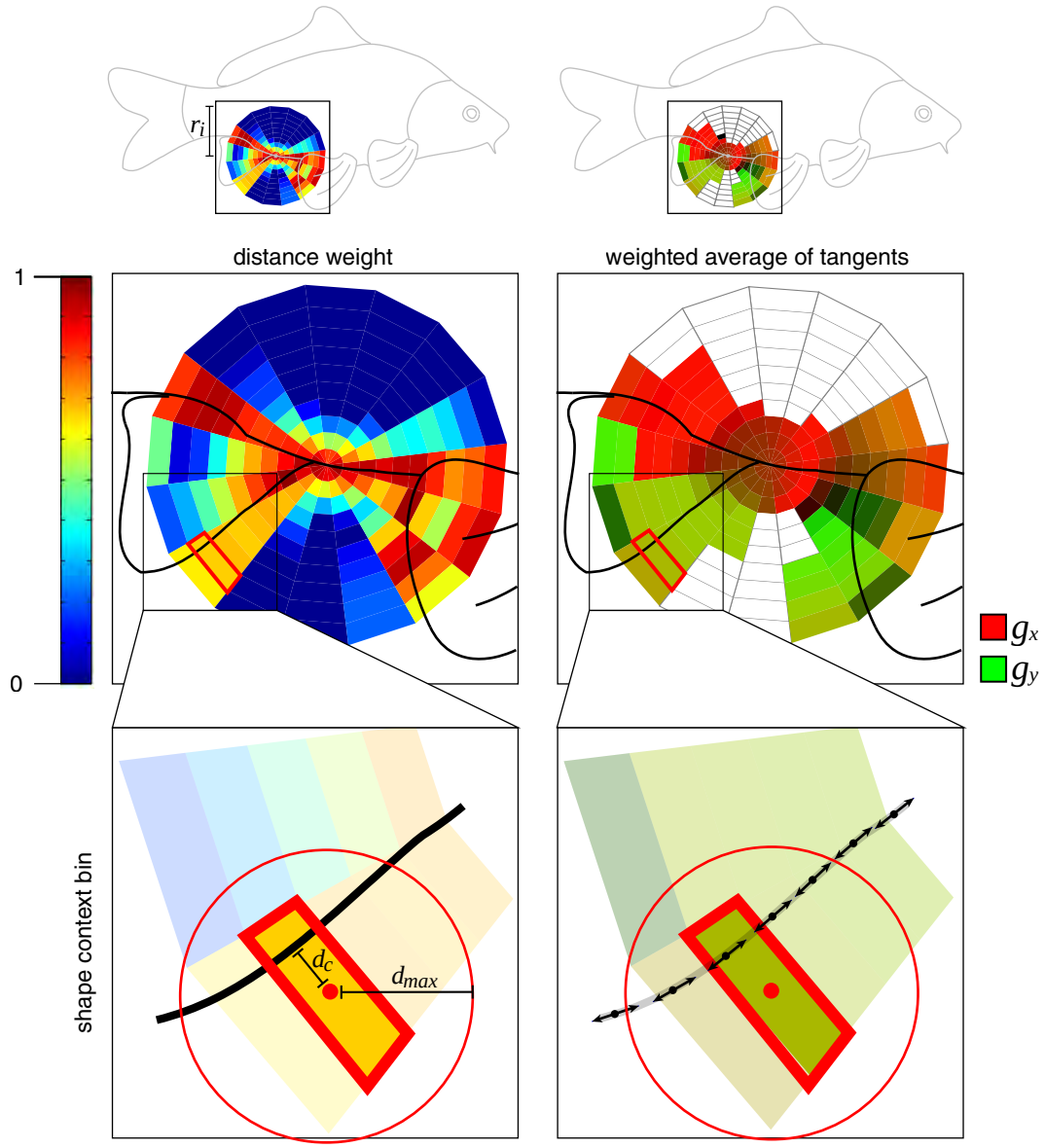


Figure A.1: We store the distance to the contours and the average contour normal in each bin of the shape contexts. The distance is shown on the left, the right side shows the average tangent per bin.

to the closest contour (see Figure A.1). The distance weight for bin k of shape context i is defined as

$$d_i^k = max(0, 1 - \frac{d_c}{d_{max}\ r_i})$$

where d_c is the distance of the bin center to the closest contour line, r_i is the radius of the shape context (i.e., the distance of the farthest bins from the center of the shape context) and d_{max} is a maximum distance defined by the user. This function models a linear falloff in a band around the

contour lines, improving the robustness of the shape context to slight variations in the positions of contour lines. The constant d_{max} controls the width of the falloff, we determined empirically that 0.3 is a good value. The tangent for a bin k is a weighted average over the tangents of all neighboring contours inside the radius d_{max}, with weights based on the distance to the bin center.

Before comparing two shape contexts, we align their frames of reference. The matching confidence for two aligned shape descriptors is then

$$f_{ij} = \frac{\sum_k d_i^k d_j^k (1 - ||g_i^k - g_j^k||^2)}{\sum_k d_i^k d_j^k},$$

where g_i^k is the average tangent in bin k of the shape context i. Note that this confidence measure is different from the original measure proposed by Belongie et al. [7]. Our measure reduces the effect of background clutter by rewarding matched bins, rather than penalizing unmatched bins. The first term $d_i^k d_j^k$ inside the sum measures the agreement of the contour line positions in both shape contexts. Note that this term is only nonzero if the contour lines are within a threshold of $2 \cdot d_{max}$. For contours that are present in one shape context, but not in the other one, the first term is zero and the sum remains unchanged. The second term measures the agreement of the contour line orientations.

All pairs of descriptors with high matching confidence can be used to form the first-order correspondences (p, τ) as described in Chapter 3, where τ are the parameters of the transformation from the local reference frame of descriptor one (including position, orientation and scale) to the local reference frame of descriptor two.

If the contours are noisy, the orientations of the shape context and consequently the rotation parameter of first-order correspondences become less reliable. Since the transformation of a correspondence is defined as $T(t_x, t_y, s, r) = T^L(t_x, t_y) * T^S(s) * T^R(r)$ (see Section 3.2), this also affects the translation parameters by an amount proportional to the distance between the correspondence and the coordinate origin. The correspondence refinement step alleviates this problem. To further improve the stability of the translation parameters in the presence of unreliable descriptor orientations, we could dynamically change the coordinate origin when computing the gradient of the translation parameters. When computing the linkage weights a_{ij} (Equation 3.6 in the paper), we can re-compute the translation parameters of both correspondences (q_i, τ_i^*) and (q_j, τ_j^*) before computing the difference quotients by moving the coordinate origin to the query point q_i of the first correspondence. The new translation parameters for both correspondences (q_*, t_*) with $* = i$ or j are then $(t_x, t_y) = (p_* - q_i) \; - \; T^S(s_*) * T^R(r_*) * (q_* - q_i)$, where $p_* = T(\tau_*) * q_*$ is the target position of the correspondence. Note that this results in an asymmetric linkage-weight matrix. To re-symmetrize it, we can take the minimum of a_{ij} and a_{ji} for each entry of the matrix.

APPENDIX B

Transformation Space Visualizations

In Figures B.1 – B.4, we show visualizations of the transformation space of several example images. In the top row of each figure, refined first-order correspondences for the scene on the left are shown in shades of gray, where darker correspondences have higher confidence. In the bottom row, we show the N best matches found by our region-matching step, the subsets of correspondences defining each matched region are shown in different colors. N is picked by the user after the regions have been found. The goal during the region-matching step of our method is to find subsets of refined correspondences that form transformation functions with low Jacobian magnitude, as shown in the bottom row. The center of the figure shows plots of the individual transformation parameters of all correspondences, while the rightmost plot shows the correspondences in three of the four transformation-space dimensions. Note how the first-order correspondences form separable clusters in transformation space (rightmost plot). Additionally, each cluster forms a transformation function with low Jacobian magnitude (see the four central plots). The caption of each figure additionally provides the inlier ratio of the first-order correspondences and the inlier ratio weighted by the confidence of the correspondences. Low numbers indicate that there is a significant amount of clutter in transformation space, suggesting that the global analysis of correspondences performed by our method is indeed necessary to identify the inliers.

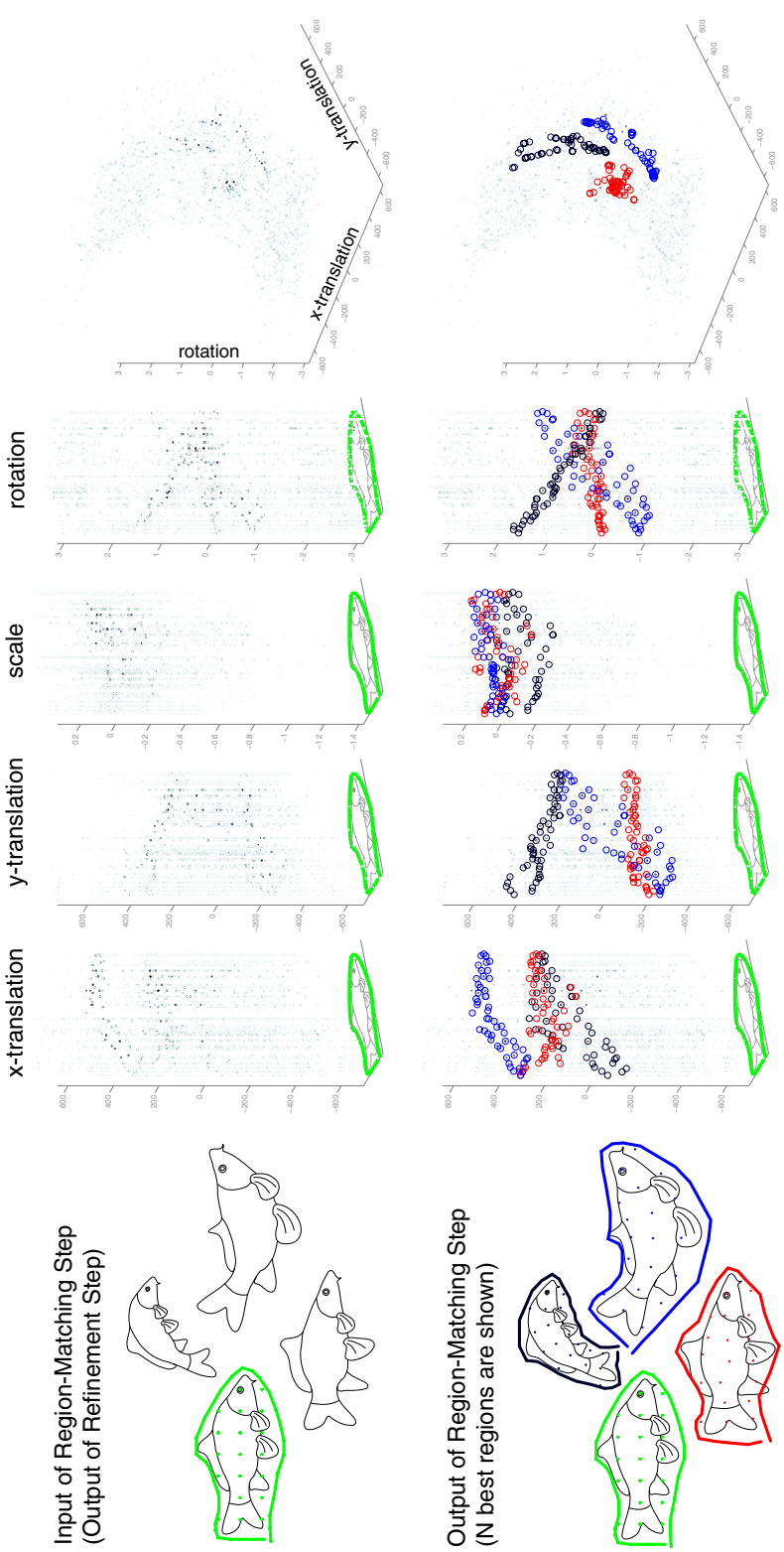


Figure B.1: A simple example where the shape of the clusters in transform space is clearly recognizable. There are a total of 5759 refined first-order correspondences and the inlier ratio is 2.62% (weighted: 12.99%).

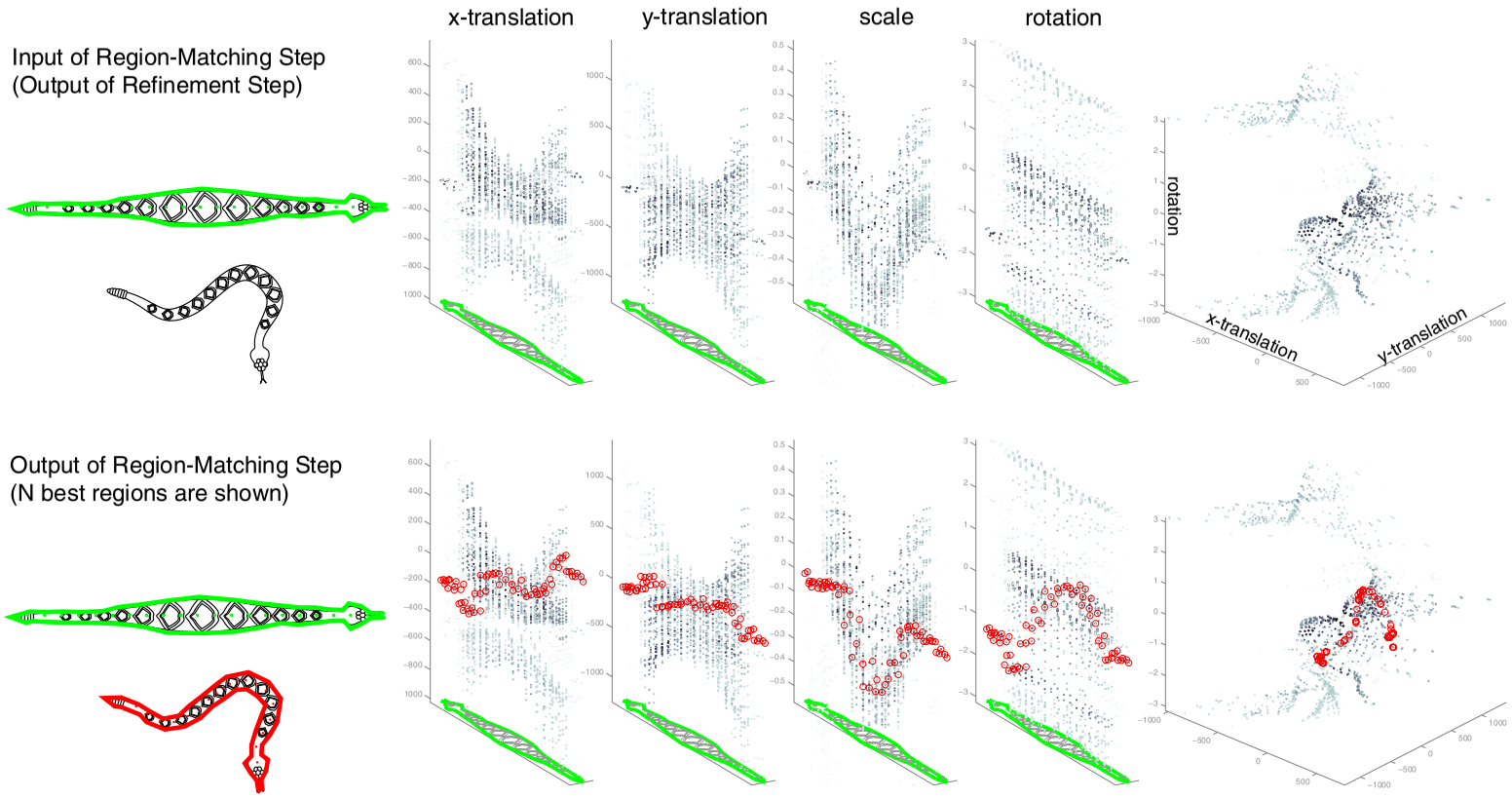


Figure B.2: The snakes image shown in Chapter 3. Note how the repeating pattern of the skin creates a lot of high-confidence outliers. There are a total of 2551 refined first-order correspondences and the inlier ratio is 2.47% (weighted: 4.18%).

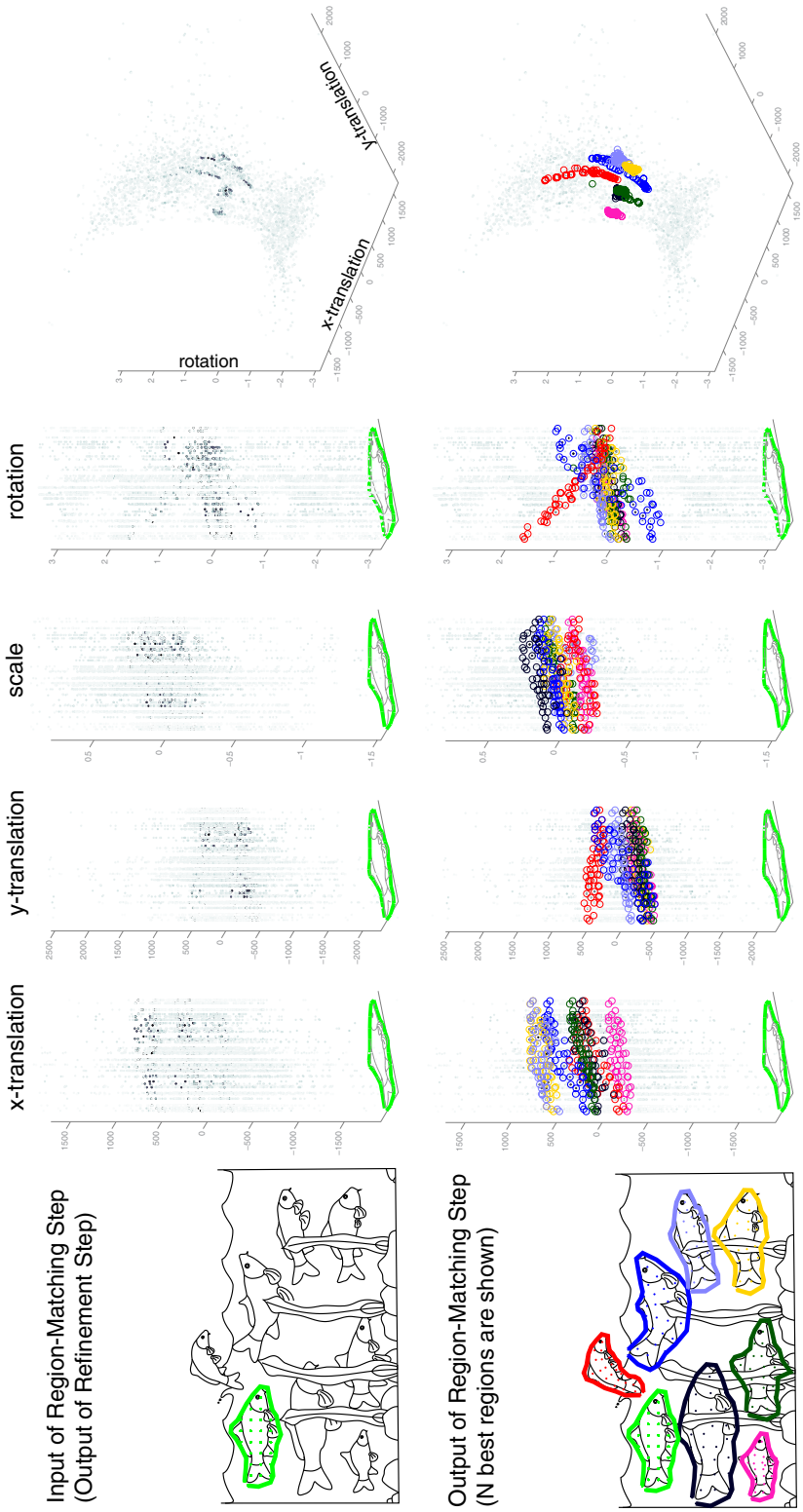


Figure B.3: The fish image shown in Chapter 3. The background clutter produces a large amount of low-confidence correspondences. Note that we cannot simply threshold the correspondences, as matched regions contain both low- and high-confidence correspondences. There are a total of 7810 refined first-order correspondences and the inlier ratio is 4.34% (weighted: 19.65%).

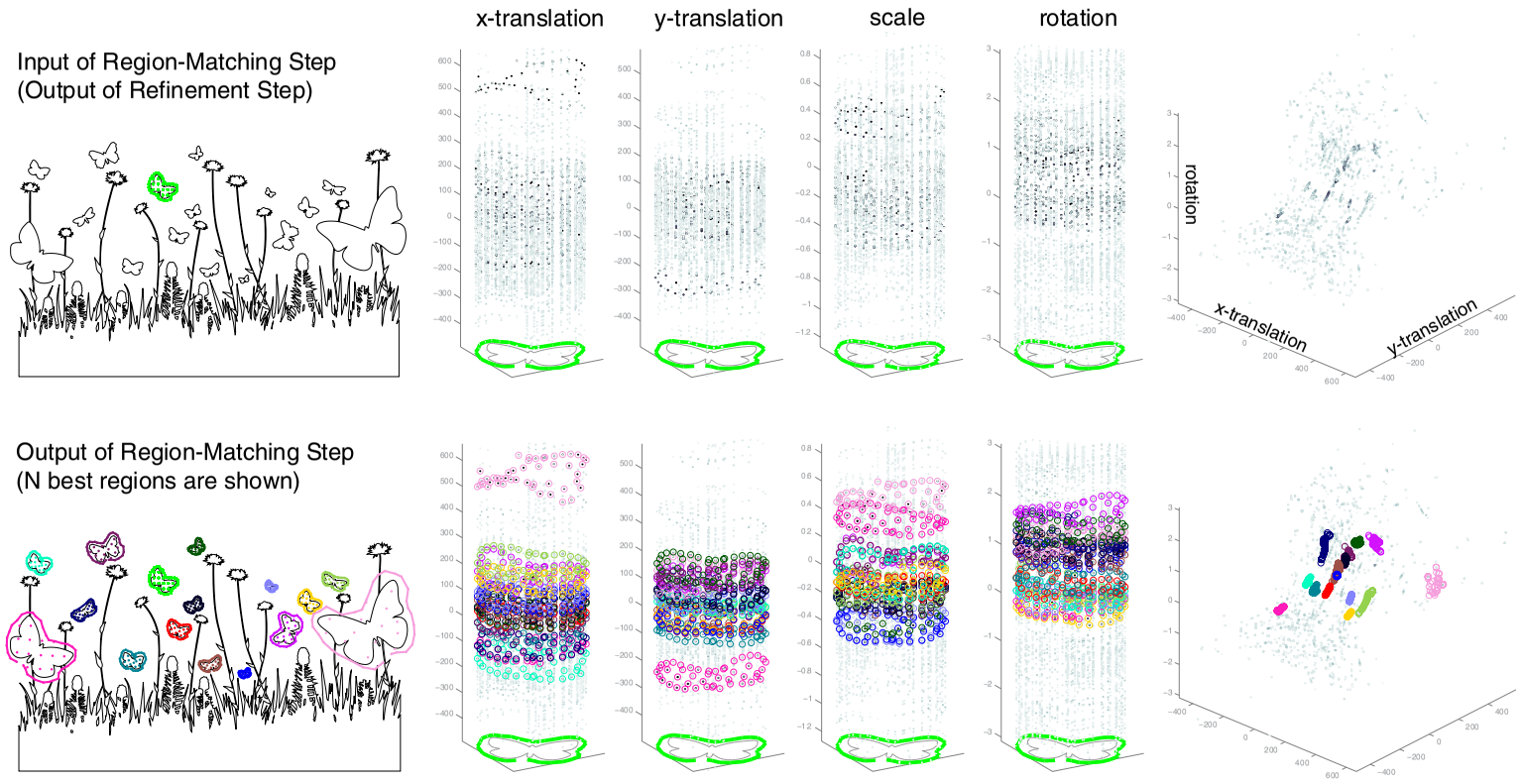


Figure B.4: The butterflies image shown in Chapter 3. Like in the fish image, the background clutter produces a large amount of low-confidence correspondences. There are a total of 5668 refined first-order correspondences and the inlier ratio is 12.35% (weighted: 29.69%).

APPENDIX C

Validity of the SL0 Kernel

C.1 Overview

In Chapter 5, a model of the user intent for placing shapes is learned from a few example placements using a kernel regression method. We use a kernel based on the smoothed ℓ_0 distance, defined in Equation 5.8. In this chapter we show that this kernel is valid. We first expand the definition of the kernel using equations (5.5), (5.4), and (5.3), and define f as the combination of assignment functions f_e for all element types. The resulting expression we call the *SL0 kernel*:

$$k_{\mathrm{SL0}}(\mathbf{x}, \mathbf{y}) = \sum_{E \in \mathbf{E}_{\mathbf{x}}} s(\mathbf{x}_E, \mathbf{y}_{f(E)}). \tag{C.1}$$

To prove this is a valid kernel, we first show that, given a set of example feature vectors $\mathbf{Q}$, the kernel evaluated between an example vector and any other feature vector is equivalent to the SL0 similarity in a high-dimensional space:

$$k_{\mathrm{SL0}}(\mathbf{q}, \mathbf{x}) = s(\mathbf{q}', \mathbf{x}') \tag{C.2}$$

for suitable vectors $\mathbf{q}'$ and $\mathbf{x}'$. Then we proceed to show that the SL0 similarity is a valid kernel.

The SL0 kernel is equivalent to an SL0 similarity in a different space. The example feature vectors $\mathbf{q} \in \mathbf{Q}$ are not part of the same vector space, since they generally do not have the same number of entries. To get the vectors into a common space, we expand them to include entries for all possible relations in the scene, i.e., the common space has dimension $n_{er} = \sum_{i=1..n_e} n_r^i$, where n_e is the number of elements in the scene and n_r^i the number of relations for the type of element i. Missing entries are filled with nil values that always result in zero SL0 similarity. We denote the expanded vectors $\widehat{\mathbf{q}}$. Feature vectors $\mathbf{x}$ are expanded using the same operation to give $\widehat{\mathbf{x}}$. Since our kernel is a sum over the SL0 similarities of individual entries, and the assignment f maximizes this sum, it is easy to see that the result of the kernel applied to the expanded feature

vectors is the same, since we have only extended the vectors by elements which are nil either in $\mathbf{q}$ or $\mathbf{x}$:

$$k_{SL0}(\mathbf{q}, \mathbf{x}) = k_{SL0}(\widehat{\mathbf{q}}, \widehat{\mathbf{x}}) = \sum_{E \in \mathbf{E}} s(\widehat{\mathbf{q}}_E, \widehat{\mathbf{x}}_{f(E)}). \tag{C.3}$$

Since both $\widehat{\mathbf{q}}$ and $\widehat{\mathbf{x}}$ are extended, their sets of corresponding elements $\mathbf{E}_{\widehat{\mathbf{q}}}$ and $\mathbf{E}_{\widehat{\mathbf{x}}}$ are both equal to the set of all elements $\mathbf{E}$. The assignment operation f can therefore be expressed as a permutation on the entries of $\widehat{\mathbf{x}}$. Since the assignment is generally different for each example vector, a different permutation is needed for each $\widehat{\mathbf{q}}$. To express the different permutations in a single vector, we append the $|\mathbf{Q}|$ permutations of $\widehat{\mathbf{x}}$ into a single vector $\mathbf{x}'$ and expand each vector $\widehat{\mathbf{q}}_i$ to have the same size as $\mathbf{x}'$, by putting $\widehat{\mathbf{q}}_i$ at the entries starting at $i * n_{er}$ and filling the remaining entries with nil. We denote the expanded example vector $\mathbf{q}'_i$. Note that this is just a reformulation of the operations performed by the kernel, and again, the result of the kernel applied to the further expanded vectors is identical to the kernel applied to the original vectors on the one hand, and equivalent to the SL0 similarity $s(\mathbf{q}', \mathbf{x}')$ on the other hand. In Section C.2, we prove this fact more formally.

SL0 similarity is a valid kernel. Next, we prove that the SL0 similarity of two vectors is a valid kernel in the sense defined by Bishop [10]. To do so, we show that the SL0 similarity can be constructed from simpler kernels that are known to be valid. The Gaussian kernel

$$k_{\mathcal{G}}(x, y) = \mathcal{G}(x - y|0, \sigma) \tag{C.4}$$

is known to be valid (see [10], Section 6.2). Then by repeatedly applying the kernel construction rule (see [10], Equation 6.21)

$$k(\mathbf{x}, \mathbf{y}) = K_a(x_a, y_a) + K_b(x_b, y_b), \tag{C.5}$$

which states that the left-hand side is a valid kernel if the kernels on the right-hand side are valid, we see that k_{SL0} is a valid kernel, q.e.d.

In Section C.3, we give a different proof of the kernel validity, which is based on a reconstruction from basis functions, and allow an interesting visualization of these basis functions.

C.2 Formal proof of validity of feature vector permutation

In this section, we prove more formally that the kernel evaluated between a vector in $\mathbf{Q}$ and any vector $\mathbf{x}$ is equivalent to the SL0 similarity of the corresponding expanded vectors $\mathbf{q}'$ and $\mathbf{x}'$.

The bijective function f that assigns elements of $\widehat{\mathbf{x}}$ to elements of $\widehat{\mathbf{q}}$ is a permutation. Let $P_{\mathbf{x},i}$ be the permutation matrix that assigns elements of $\widehat{\mathbf{x}}$ to elements of $\widehat{\mathbf{q}}_i$ and expand each row and column by the number of relationship types for the corresponding element. Zero entries are expanded with blocks of zeros, non-zero entries are expanded with identity matrices (recall that only elements of the same type are matched, so non-zero entries are always expanded with square blocks), such that

$$k_{SL0}(\mathbf{q}_i, \mathbf{x}|i) = \sum_{E \in \mathbf{E}} s(\widehat{\mathbf{q}}_{i_E}, \widehat{\mathbf{x}}_{f(E)}) = s(\widehat{\mathbf{q}}_i, P_{\mathbf{x},i}\widehat{\mathbf{x}}) \tag{C.6}$$

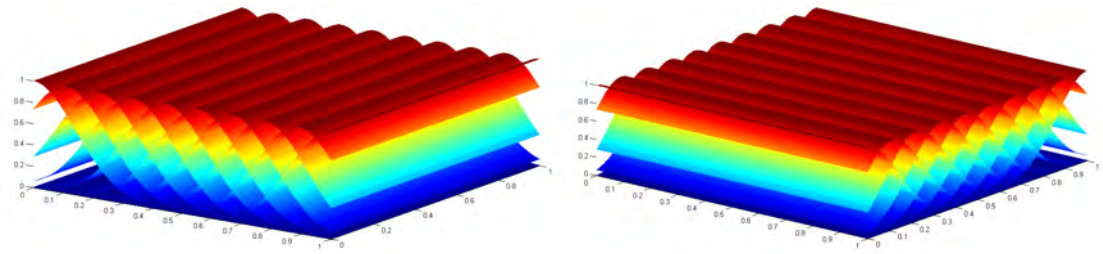


Figure C.1: Basis function for regression. We show all basis functions for a 2-dimensional feature space. The left image shows all basis functions that are non-constant in the first dimension, the right image shows the basis functions that are non-constant in the second dimension.

Note that this formulation is still dependent on i. To remove this dependency, we concatenate all permutations for vector $\widehat{\mathbf{x}}$:

$$\mathbf{x}' = P_{\mathbf{x}}\widehat{\mathbf{x}}, \tag{C.7}$$

with

$$P_{\mathbf{x}} = [P_{\mathbf{x},1}^{\mathrm{T}}, P_{\mathbf{x},2}^{\mathrm{T}}, \ldots, P_{\mathbf{x},|\mathbf{Q}|}^{\mathrm{T}}]^{\mathrm{T}}, \tag{C.8}$$

and the square brackets denote concatenation. The expansion of $\widehat{\mathbf{q}}_i$ is defined as

$$\mathbf{q}_i' = R_i\widehat{\mathbf{q}}_i, \tag{C.9}$$

with

$$R_i = [O_1, O_2, \ldots, I_i, \ldots, O_{|\mathbf{Q}|}]^{\mathrm{T}}, \tag{C.10}$$

where O_i denotes a zero matrix and I_i the identity matrix, both the same size as P_i. Then

$$k_{SL0}(\mathbf{q}, \mathbf{x}) = s(R_i\widehat{\mathbf{q}_i}, P_x\widehat{\mathbf{x}}) = s(\mathbf{q}', \mathbf{x}'), \tag{C.11}$$

q.e.d.

C.3 Alternative proof of SL0 kernel validity

In this section, we show that linear regression with SL0 similarity as kernel (and thus, as shown before, also with the SL0 kernel) is the equivalent dual representation of a linear regression model that uses a regularly spaced set of extruded 1D gaussian basis functions (see Figure C.1) as the number of basis functions approaches infinity. The model is defined as:

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{M} w_j\phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) \tag{C.12}$$

where M is the number of basis functions and the basis functions ϕ_j have the shape described above:

$$\phi_j(\mathbf{x}) = \mathcal{G}(x_{k(j)}|\mu_j, \sigma) \tag{C.13}$$

where $k(j)$ is the non-constant dimension of basis function j. We assume a constant variance σ, so the basis functions can be parametrized by their mean and the index of their non-constant dimension k. To enable the use of a fixed, finite set of basis functions, we restrict the domain of our model to a hyper-rectangle in feature space, defined by a minimum and maximum value in each dimension. Along each of the dimensions, we place a fixed number L_k of basis functions, for a total of $M = \sum_k L_k$ basis functions in our model. For dimension k we place basis functions that are non-constant in that dimension in regular intervals $\mu_{k1} \ \ldots \ \mu_{kL}$ between the minimum and the maximum value of that dimension. Equation C.12 then becomes:

$$y(\mathbf{x}, \mathbf{w}) = \sum_{k=1}^{D} \sum_{l=1}^{L_k} w_{kl} \, \mathcal{G}(x_k | \mu_{kl}, \sigma) \tag{C.14}$$

To show that this model is equivalent to linear regression with the SL0 kernel, we show that (see [10], Equation 6.10):

$$k_{\mathrm{SL0}}(\mathbf{x}, \mathbf{y}) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{y}) = \sum_{j=1}^{M} \phi_j(\mathbf{x}) \phi_j(\mathbf{y}) \tag{C.15}$$

as M approaches infinity. Using Equation C.13, this becomes:

$$k_{\mathrm{SL0}}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{M} \mathcal{G}(x_{k(j)} | \mu_j, \rho) \, \mathcal{G}(y_{k(j)} | \mu_j, \rho) \tag{C.16}$$

where we use basis functions with variance ρ. Recall that the basis functions are defined on regular intervals μ_{kl} in each dimension of the feature space. We can therefore split the sum into an outer sum over all dimensions and an inner sum over all basis functions in that dimension.

$$k_{\mathrm{SL0}}(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{D} \sum_{l=1}^{L_k} \mathcal{G}(x_k | \mu_{kl}, \rho) \, \mathcal{G}(y_k | \mu_{kl}, \rho) \tag{C.17}$$

Since the Gaussian function is symmetric around the origin, we can switch the argument with the parameter μ

$$k_{\mathrm{SL0}}(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{D} \sum_{l=1}^{L_k} \mathcal{G}(\mu_{kl} | x_k, \rho) \, \mathcal{G}(\mu_{kl} | y_k, \rho). \tag{C.18}$$

The scaling of the Gaussians is unimportant. When scaling a Gaussian by a factor c, the result of the regression is the same (the corresponding entry in $\mathbf{w}$ is only scaled by the inverse c^{-1} of that factor). We can scale the Gaussians by the distance $\Delta\mu_k$ between two adjacent Gaussian centers. Note that $\Delta\mu_k$ only depends on the dimension since the centers of the Gaussians are spaced equally along each dimension.

$$k_{\mathrm{SL0}}(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{D} \sum_{l=1}^{L_k} \mathcal{G}(\mu_{kl} | x_k, \rho) \, \mathcal{G}(\mu_{kl} | y_k, \rho) \, \Delta\mu_k \tag{C.19}$$

As L_k goes to infinity, the sum becomes an integral. Factoring out the inverse normalization factor $\frac{1}{\sqrt{2\pi\rho}}$ for a normal distribution from each Gaussian we get

$$k_{\mathrm{SL0}}(\mathbf{x}, \mathbf{y}) = 2\pi\rho \sum_{k=1}^{D} \int \mathcal{N}(\mu_k | x_k, \rho)\, \mathcal{N}(\mu_k | y_k, \rho)\, d\mu_k \tag{C.20}$$

The integral of the product of two normal distributions is a zero-mean normal distribution of the distance between the centers:

$$k_{\mathrm{SL0}}(\mathbf{x}, \mathbf{y}) = 2\pi\rho \sum_{k=1}^{D} \mathcal{N}(x_k - y_k | 0, 2\rho) \tag{C.21}$$

$$= \sqrt{\pi\rho} \sum_{k=1}^{D} \mathcal{G}(x_k - y_k | 0, 2\rho) \tag{C.22}$$

Which is equivalent to the definition of the SL0 kernel with $\sigma = 2\rho$ up to the constant scaling factor $\sqrt{\pi\rho}$, q.e.d.

Bibliography

[1] Dragomir Anguelov, Praveen Srinivasan, Hoi-Cheung Pang, Daphne Koller, Sebastian Thrun, and James Davis. The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces. *Adv. in Neural Inf. Proc. Systems*, 17:33–40, 2005.

[2] Haim Avron, Andrei Sharf, Chen Greif, and Daniel Cohen-Or. L1-sparse reconstruction of sharp point set surfaces. *ACM Trans. Graph.*, 29(5):135:1–135:12, November 2010.

[3] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, 2004.

[4] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch. *ACM TOG*, 28:24:1–24:11, 2009.

[5] Connelly Barnes, Eli Shechtman, Dan B. Goldman, and Adam Finkelstein. The generalized patchmatch correspondence algorithm. ECCV'10, pages 29–43, 2010.

[6] Evan Behar and Jyh-Ming Lien. Fast and robust 2d minkowski sum using reduced convolution. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, San Francisco, CA, Sep. 2011.

[7] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(4):509–522, 2002.

[8] Alexander Berner, Michael Wand, Niloy J. Mitra, Daniel Mewes, and Hans-Peter Seidel. Shape analysis with subspace symmetries. *CGF*, 30(2), 2011. Eurographics.

[9] Floraine Berthouzoz, Wilmot Li, Mira Dontcheva, and Maneesh Agrawala. A framework for content-adaptive photo manipulation macros: Application to face, landscape, and global manipulations. *ACM Trans. Graph.*, 30(5):120:1–120:14, October 2011.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[11] Martin Bokeloh, Michael Wand, Vladlen Koltun, and Hans-Peter Seidel. Pattern-aware shape deformation using sliding dockers. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 123:1–123:10, 2011.

[12] Martin Bokeloh, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun. An algebraic model for parameterized shape editing. *ACM Trans. Graph.*, 31(4):78:1–78:10, July 2012.

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

[14] Mario Botsch, Mark Pauly, Markus Gross, and Leif Kobbelt. Primo: Coupled prisms for intuitive surface modeling. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 11–20, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[15] Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. Generalized multidimensional scaling: A framework for isometry-invariant partial surface matching. *Proceedings of the National Academy of Sciences of the United States of America*, 103(5):1168–1172, 2006.

[16] Will Chang and Matthias Zwicker. Automatic registration for articulated shapes. In *SGP*, pages 1459–1468, 2008.

[17] Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. Probabilistic reasoning for assembly-based 3d modeling. *ACM Trans. Graph.*, 30(4):35:1–35:10, July 2011.

[18] Ming-Ming Cheng, Fang-Lue Zhang, Niloy J. Mitra, Xiaolei Huang, and Shi-Min Hu. Repfinder: finding approximately repeated scene elements for image editing. *ACM TOG*, 29:83:1–83:8, 2010.

[19] Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Feature correspondence and deformable object matching via agglomerative correspondence clustering. In *ICCV*, pages 1280–1287, 2009.

[20] Haili Chui and Anand Rangarajan. A new point matching algorithm for non-rigid registration. *Comput. Vis. Image Underst.*, 89(2-3):114–141, 2003.

[21] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE PAMI*, 24(5):603–619, 2002.

[22] M. Cui, J. Femiani, J. Hu, P. Wonka, and A. Razdan. Curve matching for open 2d curves. *Pattern Recogn. Lett.*, 30(1):1–10, 2009.

[23] Zhifu Cui, Hang Zhang, and Wei Lu. An improved smoothed l0-norm algorithm based on multiparameter approximation function. In *Communication Technology (ICCT), 2010 12th IEEE International Conference on*, pages 942–945, 2010.

[24] A Elad and R. Kimmel. Bending invariant representations for surfaces. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–168–I–174 vol.1, 2001.

[25] A. Elad and R. Kimmel. On bending invariant signatures for surfaces. *IEEE PAMI*, 25(10):1285 – 1295, 2003.

[26] Vittorio Ferrari, Frederic Jurie, and Cordelia Schmid. From images to shape models for object detection. *Int. J. Comput. Vision*, 87(3):284–303, May 2010.

[27] Matthew Fisher and Pat Hanrahan. Context-based search for 3d models. *ACM Trans. Graph.*, 29(6):182:1–182:10, December 2010.

[28] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based synthesis of 3d object arrangements. *ACM Trans. Graph.*, 31(6):135:1–135:11, November 2012.

[29] Matthew Fisher, Manolis Savva, and Pat Hanrahan. Characterizing structural relationships in scenes using graph kernels. *ACM Trans. Graph.*, 30(4):34:1–34:12, July 2011.

[30] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 950–953, New York, NY, USA, 2003. ACM.

[31] Michael S. Floater. Mean value coordinates. *Comput. Aided Geom. Des.*, 20(1):19–27, March 2003.

[32] Michael S. Floater, Géza Kós, and Martin Reimers. Mean value coordinates in 3d. *CAGD*, 22:623–631, 2005.

[33] Michel Foucault. *The Order of Things: An Archaeology of the Human Sciences*. Pantheon Books, 1970.

[34] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *ACM Trans. Graph.*, 23(3):652–663, 2004.

[35] Ran Gal and Daniel Cohen-Or. Salient geometric features for partial shape matching and similarity. *ACM TOG*, 25(1):130–150, 2006.

[36] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. iWIRES: an analyze-and-edit approach to shape manipulation. *ACM Trans. Graph.*, 28(3):33:1–33:10, July 2009.

[37] Natasha Gelfand and Leonidas J. Guibas. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pages 214–223, New York, NY, USA, 2004. ACM.

[38] Pijush K. Ghosh. A solution of polygon containment, spatial planning, and other related problems using minkowski operations. *Comput. Vision Graph. Image Process.*, 49(1):1–35, January 1990.

[39] J. Gil and M. Werman. Computing 2-d min, median, and max filters. *PAMI, IEEE Transactions on*, 15(5):504 –507, may 1993.

[40] K. Gębal, J. A. Bærentzen, H. Aanæs, and R. Larsen. Shape analysis using the auto diffusion function. In *Proceedings of the Symposium on Geometry Processing*, SGP '09, pages 1405–1413. Eurographics Association, 2009.

[41] Mikhail Gromov. *Metric Structures for Riemannian and Non-Riemannian Spaces*. Birkhäuser, 1999.

[42] Paul Guerrero, Stefan Jeschke, Michael Wimmer, and Peter Wonka. Edit propagation using geometric relationship functions. *Transactions on Graphics*, 2014.

[43] Yoav HaCohen, Eli Shechtman, Dan B Goldman, and Dani Lischinski. Non-rigid dense correspondence with applications for image enhancement. *ACM TOG*, 30(4):70:1–70:9, 2011.

[44] Kai Hormann and Michael S. Floater. Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph.*, 25(4):1424–1441, October 2006.

[45] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM TOG*, 30(4):78:1–78:8, 2011.

[46] Andrew Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1997.

[47] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26(3), July 2007.

[48] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, July 2005.

[49] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph.*, 31(4):55:1–55:11, July 2012.

[50] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3d mesh segmentation and labeling. *ACM Trans. Graph.*, 29(4):102:1–102:12, July 2010.

[51] Rosenfeld A. Kirchen L. Gray-level corner detection. *Pattern Recognition Letters 1*, 1982.

[52] Vladislav Kraevoy and Alla Sheffer. Mean-value geometry encoding. *International Journal of Shape Modeling*, 12(01):29–46, 2006.

[53] Vladislav Kraevoy, Alla Sheffer, Ariel Shamir, and Daniel Cohen-Or. Non-homogeneous resizing of complex models. *ACM Trans. Graph.*, 27(5):111:1–111:9, December 2008.

[54] D. Kroon. Shape context based corresponding point models. Matlab File Exchange, 2011.

[55] Joseph Kruskal and Myron Wish. Multidimensional scaling. In *University Paper series on Quantitative Application in Social Sciences*, volume 11, 1978.

[56] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:86–97, 1955.

[57] Torsten Langer, Alexander Belyaev, and Hans-Peter Seidel. Spherical barycentric coordinates. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 81–88, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[58] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *ICCV*, volume 2, pages 1482–1489 Vol. 2, 2005.

[59] Yaron Lipman and Thomas Funkhouser. Mobius voting for surface correspondence. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), aug 2009.

[60] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27(3):78:1–78:10, August 2008.

[61] Markus Lipp, Daniel Scherzer, Peter Wonka, and Michael Wimmer. Interactive modeling of city layouts using layers of procedural content. *Computer Graphics Forum (Proceedings EG 2011)*, 30(2):345–354, April 2011.

[62] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics*, 27(3):102:1–10, August 2008. Article No. 102.

[63] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. C. Vision*, 60:91–110, 2004.

[64] ChengEn Lu, Longin Jan Latecki, Nagesh Adluru, Xingwei Yang, and Haibin Ling. Shape guided contour grouping with particle filters. In *ICCV*, pages 2288–2295. IEEE, 2009.

[65] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, 1981.

[66] Jiayi Ma, Ji Zhao, Jinwen Tian, Zhuowen Tu, and A.L. Yuille. Robust estimation of nonrigid transformation for point set registration. pages 2147–2154, 2013.

[67] Tianyang Ma and L.J. Latecki. From partial shape matching through local deformation to robust global shape similarity for object detection. In *CVPR*, pages 1441 –1448, 2011.

[68] Hiroshi Masuda and Kenta Ogawa. Application of interactive deformation to assembled mesh models for cae analysis. *ASME Int. Design Engineering Technical Conferences*, 2007.

[69] Hiroshi Masuda, Yasuhiro Yoshioka, and Yoshiyuki Furukawa. Preserving form features in interactive mesh deformation. *Comput. Aided Des.*, 39(5):361–368, May 2007.

[70] Paul Merrell. Example-based model synthesis. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 105–112, New York, NY, USA, 2007. ACM.

[71] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph.*, 30(4):87:1–87:10, July 2011.

[72] Niloy J. Mitra, Leonidas Guibas, and Mark Pauly. Symmetrization. *ACM TOG*, 26(3):1–8, 2007.

[73] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM TOG*, 25:560–568, 2006.

[74] Niloy J. Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3d geometry: Extraction and applications. In *EUROGRAPHICS State-of-the-art Report*, 2012.

[75] Facundo Mémoli and Guillermo Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Found. Comput. Math.*, 5:313–347, July 2005.

[76] G. Hosein Mohimani, Massoud Babaie-Zadeh, and Christian Jutten. A fast approach for overcomplete sparse decomposition based on smoothed l0 norm. *CoRR*, abs/0809.2508, 2008.

[77] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: A flexible representation of maps between shapes. *ACM Trans. Graph.*, 31(4):30:1–30:11, July 2012.

[78] Christian Schou Oxvig, Patrick Steffen Pedersen, Thomas Arildsen, and Torben Larsen. Improving smoothed l0 norm in compressive sensing using adaptive parameter selection. *CoRR*, abs/1210.4277, 2012.

[79] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.

[80] G. Patow. User-friendly graph editing for procedural modeling of buildings. *Computer Graphics and Applications, IEEE*, 32(2):66–75, March 2012.

[81] J. Pokrass, A. M. Bronstein, M. M. Bronstein, P. Sprechmann, and G. Sapiro. Sparse modeling of intrinsic correspondences. *CGF*, 32(2pt4):459–468, 2013.

[82] Tiberiu Popa, Dan Julius, and Alla Sheffer. Interactive and linear material aware deformations. *International Journal of Shape modeling*, 2007.

[83] Anand Rangarajan, Haili Chui, and FredL. Bookstein. The softassign procrustes matching algorithm. In *Information Processing in Medical Imaging*, volume 1230 of *Lecture Notes in Computer Science*, pages 29–42. Springer Berlin Heidelberg, 1997.

[84] Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke. Laplace-beltrami spectra as "shape-dna" of surfaces and solids. *Computer-Aided Design*, 38(4):342–366, 2006.

[85] Azriel Rosenfeld, Robert A Hummel, and S.W. Zucker. Scene labeling by relaxation operations. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(6):420–433, June 1976.

[86] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[87] Pedro Silva, Pascal Müller, Rafael Bidarra, and Antonio Coelho. Node-based shape grammar representation and editing. In *Proceedings of PCG 2013*, Chania, Crete, Greece, may 2013.

[88] Peter-Pike J. Sloan, Charles F. Rose, III, and Michael F. Cohen. Shape by example. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, pages 135–143, New York, NY, USA, 2001. ACM.

[89] Ruben M. Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum*, 2014.

[90] Justin Solomon, Andy Nguyen, Adrian Butscher, Mirela Ben-Chen, and Leonidas Guibas. Soft maps between surfaces. *Comp. Graph. Forum*, 31(5):1617–1626, August 2012.

[91] Praveen Srinivasan, Qihui Zhu, and Jianbo Shi. Many-to-one contour matching for describing and discriminating object shape. In *IEEE CVPR*, pages 1673–1680, 2010.

[92] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *SGP*, pages 1383–1392, 2009.

[93] Tim Tutenel, Ruben Michaël Smelik, Rafael Bidarra, and Klaas Jan de Kraker. A semantic scene description language for procedural layout solving problems. In G. Michael Youngblood and Vadim Bulitko, editors, *AIIDE*. The AAAI Press, 2010.

[94] Oliver van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. A survey on shape correspondence. *Computer Graphics Forum*, 30(6):1681–1707, 2011.

[95] B. Watson, P. Muller, P. Wonka, C. Sexton, O. Veryovka, and A Fuller. Procedural urban modeling in practice. *Computer Graphics and Applications, IEEE*, 28(3):18–26, May 2008.

[96] Xie Xiao-Chun. Compressive sensing radar imaging based on smoothed l0 norm. In *Signal Processing Systems (ICSPS), 2010 2nd International Conference on*, volume 1, pages V1–6–V1–9, 2010.

[97] Ken Xu and et al. Constraint-based automatic placement for scene composition. In *GRAPHICS INTERFACE*, pages 25–34, 2002.

[98] Weiwei Xu, Jun Wang, KangKang Yin, Kun Zhou, Michiel van de Panne, Falai Chen, and Baining Guo. Joint-aware manipulation of deformable models. *ACM Trans. Graph.*, 28(3):35:1–35:9, July 2009.

[99] Jun Ye and Zhong Jin. Nonnegative matrix factorization on orthogonal subspace with smoothed l0 norm constrained. 7751:1–7, 2013.

[100] Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D. Goodman, and Pat Hanrahan. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM TOG*, 31(4):56:1–56:11, July 2012.

[101] Lap-Fai Yu, Sai-Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F. Chan, and Stanley J. Osher. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.*, 30(4):86:1–86:12, July 2011.

[102] Kaan Yücer, Alec Jacobson, Alexander Hornung, and Olga Sorkine. Transfusive image manipulation. *ACM TOG*, 31(6):176:1–176:9, 2012.

[103] Hao Zhang, Alla Sheffer, Daniel Cohen-Or, Qingnan Zhou, Oliver van Kaick, and Andrea Tagliasacchi. Deformation-driven shape correspondence. *SGP*, 27(5):1431–1439, 2008.

[104] Yefeng Zheng and D. Doermann. Robust point matching for nonrigid shapes by preserving local neighborhood structures. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):643–649, 2006.

[105] Youyi Zheng, Xiang Chen, Ming-Ming Cheng, Kun Zhou, Shi-Min Hu, and Niloy J. Mitra. Interactive images: cuboid proxies for smart image manipulation. *ACM Trans. Graph.*, 31(4):99:1–99:11, July 2012.

[106] Youyi Zheng, Hongbo Fu, Daniel Cohen-Or, Oscar Kin-Chung Au, and Chiew-Lan Tai. Component-wise controllers for structure-preserving shape manipulation. *Computer Graphics Forum*, 30(2):563–572, 2011.

Curriculum Vitae

Personal Information

Name:	Paul Guerrero
Academic Degree:	Dipl.-Ing. (equiv. M.Sc.)
Address:	Sanatoriumstr. 21b 17/3 1140 Vienna, Austria
Email:	paul.guerrero@chello.at, paul@cg.tuwien.ac.at
Date of birth:	June 10th, 1981 in Vienna, Austria
Marital Status:	Single
Languages:	German, English, Spanish

Education

since May 2008	PhD candidate at Vienna University of Technology, Department of Computer Graphics and Algorithms
2000 – 2007	Studies in Computer Science at Vienna University of Technology, Austria, with special emphasis on Computer Graphics Graduation as "Diplom-Ingenieur" from Vienna University of Technology (Thesis: "Approximative Real-time Soft Shadows and Diffuse Reflections in Dynamic Scenes") in October 2007
December 2005	Participation in the "2005 UCSB International Capture The Flag" (iCTF) Internet Security Contest, team got 2nd place (best defensive performance)
1991 – 2000	Secondary School (Gymnasium) in Burghausen (Germany), Albany, N.Y. (USA) and Bogotá (Colombia) Colombian Graduation (ICFES) in 1999 and German Graduation (Abitur) in 2000

Professional

November 2013 – June 2014	Visiting PhD student at KAUST, Saudi Arabia
October 2012 – June 2013	Visiting PhD student at KAUST, Saudi Arabia
July 2010	Lecturer at the Summer School 2010 at the UCI in Cuba
October 2009 – June 2011	Organizer of a Seminar on Methodical Working and Lecturer at a Seminar on Computer Graphics at Vienna University of Technology
October 2009	Co-Organizer of a Joint Seminar on Visual Computing in Moscow
May 2008 – May 2012	University Assistant at Vienna University of Technology, Department of Computer Graphics and Algorithms
March 2006 – Oct 2006	Internship at Vienna University of Technology, Department of Computer Graphics and Algorithms
July 2005 – March 2006	Internship at Vienna University of Technology, Department of Pattern Recognition and Image Processing
2004 – 2006	Work on a freeware game (maxfighter.musgit.com), released under the GNU General Public License
1996 – 2007	Design and creation of various webpages (including pcmegamedia.com, casaalemana.net, polo-austria.at)

Publications

Paul Guerrero, Stefan Jeschke, Michael Wimmer, Peter Wonka. Edit Propagation using Geometric Relationship Functions. *ACM Trans. Graph.*, 33, 2, Article 15, 2014.

Paul Guerrero, Thomas Auzinger, Michael Wimmer, Stefan Jeschke. Partial Shape Matching using Transformation Parameter Similarity. *Computer Graphics Forum. To be published.*

Paul Guerrero, Stefan Jeschke, Michael Wimmer, Real-time indirect illumination and soft shadows in dynamic scenes using spherical lights. *Computer Graphics Forum*, 27, 8, pages 2154–2168, 2008.