Model repair and editing tools

by

Vladislav Kraevoy

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

June, 2007

Abstract

With the declining production cost and improvement of scanning technology, three-dimensional model acquisition systems are rapidly becoming more affordable. At the same time, personal computers with graphics hardware capable of displaying complex 3D models have become inexpensive enough to be available to a large population. As a result, there is, potentially, an opportunity to consider new virtual reality uses from areas as diverse as cultural heritage exploration and retail sales applications that will allow people to view associated large classes of realistic 3D objects on home computers and media devices. Although there are many physical techniques for acquiring 3D data, including laser scanners, CT or MRI scans, the basic pipeline of operations (Figure 1.1) lacks a sufficient set of tools to take the acquired data and produce a usable 3D model.

This dissertation proposes a set of efficient and robust 3D data reconstruction and editing tools for such a pipeline. We look at the fundamental problems of range scan data completion, modeling, and parameterization. We propose a new cross-parameterization method for efficient calculation of a low-distortion bijective mapping between models. Recent research in digital geometry processing suggests multiple new applications for such a mapping, including pair-wise model editing [11] transferring texture and surface properties (BRDFs, normal maps, etc) [61], and fitting template meshes to multiple data sets [7, 55]. We also extend our cross-parameterization technique to support models with gaps and holes. This allows us to develop a new and robust method for template-based range scan data completion.

One of the most significant obstacles in computer graphics is providing easy-to-use tools for creating and editing detailed 3D models. To this end, we present a new set of tools with which non-expert user can create detailed geometric models quickly and easily. In particular, we propose a new modeling system for creating new, original

models by mixing and matching parts of pre-existing models. In this way, we eliminate the need for a user to perform complex geometric operations, and thereby reduce the modeling process to that of part selection.

This dissertation also proposes a new technique for image-based modeling that allows a user to easily transform a sketch or picture into a 3D model using a 3D template model. The 3D template provides the geometric detail that cannot be inferred from an image alone. This allows the user to create detailed geometric models from pictures alone.

We also introduce a real-time editing algorithm that allows the creation of new models through the deformation of existing ones. Our proposed editing algorithm has applications in such common geometric operations as mesh deformation, morphing, and blending. Thus, we propose contributions to the model repair and editing pipeline that simplifies the task of creating and repairing detailed 3D models.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Assist. Prof. Alla Sheffer, whose expertise, understanding, and patience added considerably to my graduate experience. I also thank the members of my supervisory and examining committees Assoc. Prof. Wolfgang Heidrich, Assoc. Prof. Michiel van de Panne, Prof. Yusuf Altintas, Prof. Uri Ascher and Prof. Markus Gross for taking time out from their busy schedules to review this thesis and for their valuable suggestions.

A special thanks goes out to Prof. Craig Gotsman , without whose motivation and encouragement I would not have considered a graduate career in computer graphics.

My thanks to my friends and colleagues for the great time I had at UBC. I enjoyed the atmosphere, their friendship, and their support. Abhijeet Ghosh, David Burke, Kristian Hildebrand, Ciaran Llachlan Leavitt, Fred Kimberley, James Slack, Chen Yang, Matthew Trentacoste, Cheryl Lau, Dan Julius, Kenneth Rose and Tiberiu Popa made Imager a really fun place to be.

My thanks to Abhijeet Ghosh for his collaboration while writing his own doctoral thesis.

My thanks to Dan Julius for the collaboration on several papers.

My thanks to Ciaran Llachlan Leavitt for proof-reading several papers and for providing helpful suggestions for improving this manuscript. Any 'linguistic crimes' are mine alone.

I recognize that this research would not have been possible without the financial assistance. This dissertation work was supported by a UBC University Graduate Entrance Scholarship (2003-04) and a UBC University Graduate Fellowship (2005-07).

Last but not least, I wish to thank my parents Grigory and Galina Kraevoy who have always supported me, and without whom none of this would have been even possible. This thesis is dedicated to them.

Chapter 1

Introduction

One of the major challenges in geometry processing is to provide easy-to-use tools for the creation and manipulation of digital models. Although there are many physical techniques for acquiring 3D data, including laser scanners, CT and MRI scans, the basic pipeline of operations lacks a sufficient set of tools to take the acquired data and produce a complete animated surface model ready for rendering. This dissertation proposes a set of efficient and robust 3D data parameterization and editing tools for such a pipeline (Figure 1.1). We investigate different algorithms, and applications of mesh parameterization such as: morphing; shape blending; texture and material properties transfer; and fitting template meshes to scan data. We also present a set of new modeling tools for novel 3D content creation through the manipulation of one or more pre-existing models. These tools include techniques for model deformation along with model composition and image-based modeling, among others that will be discussed in the following sections in greater detail.

1.1 Parameterization

Many geometry processing applications, such as morphing, shape blending, transferring texture or material properties, and fitting template meshes to scan data, require a bijective mapping between two or more models. The models that need to be mapped, that is cross-parameterized, usually have similar features (there is little use for mapping a phone onto a cow) and the mapping must respect those. For example, when mapping between two humans, the legs must map to the legs, the ears to the ears, and so on. This is typically achieved by enforcing the correspondence for a small set of feature vertices and using a cross-parameterization that preserves the shape of the models as

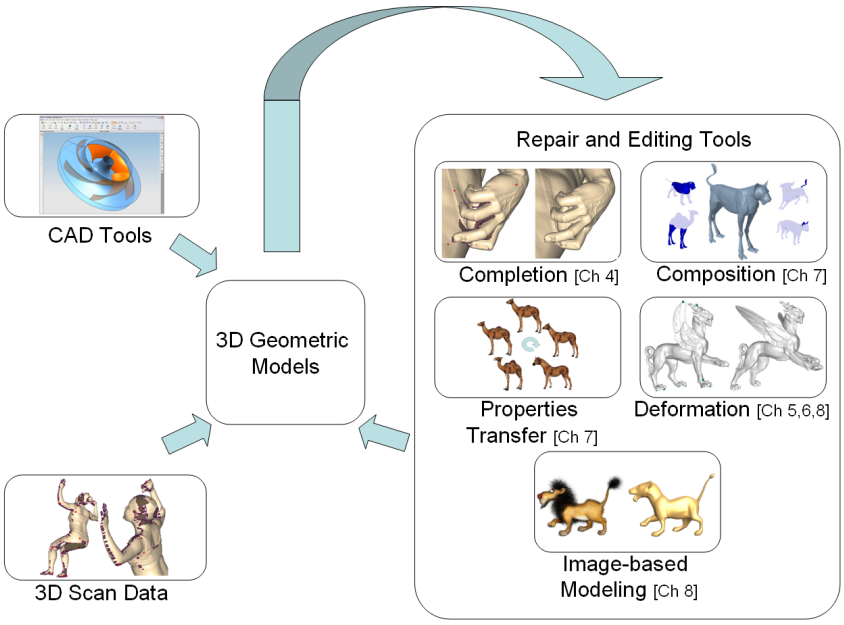


Figure 1.1: General overview of model repair and editing pipeline.

much as possible.Most of the applications, such as blending and morphing, require compatible meshes, i.e. meshes with identical connectivity. Therefore, given the cross-parameterization, the models need to be remeshed with a common connectivity.

In this dissertation we propose new methods for cross-parameterization and compatible remeshing. Our cross-parameterization method [42] is the first one guaranteed to find a bijective parameterization that satisfies the feature vertex correspondence between any number of models without additional user input. We propose a novel local framework for mesh parameterization over a base mesh domain. Our algorithm uses this framework to compute both shape-preserving and adaptive cross-parameterizations. The framework can be used to optimize other criteria, providing a powerful stand-alone parameterization tool. After the cross-parameterization is computed, a novel adaptive method for compatible remeshing is applied. It generates meshes that accurately approximate the input geometry with significantly fewer elements than those generated by previous techniques. As a consequence, the method can

be applied to generate compatible meshes for any number of models. Thanks to the combination of shape preservation and good approximation, the resulting compatible meshes are well-suited for morphing and other geometry processing applications.

Models generated by range scanners and other acquisition tools are often incomplete and typically contain multiple connected components with irregular boundaries and complex holes. Readily available commercial software is used to merge multiple scans. Unfortunately, the resulting merged scans remain incomplete because of occlusions and grazing angle views. Scanners also often fail to capture complex model features such as toes or ears. When performing scan completion, the difficulty is to correctly reconstruct the connectivity and the topology as well as to complete the missing geometric details. To correctly repair such models global shape information, such as the one provided by a template, is necessary. We extend our cross-parameterization technique to support models with gaps and holes. This allows us to develop a new and robust method for template-based range scan data reconstruction [43]. Our new work introduces a robust algorithm for completion of such data by using a cross-parameterization between the incomplete model and a template model provided by the user. We employ this parameterization to correctly glue together the parts of the input model and to close the holes. The template model is also used to fill in the topological and geometric information missing in the input. This extended cross-parameterization mechanism is more generic than previous techniques; it guarantees bijectivity and introduces less distortion.

1.2 Modeling

The creation of novel 3D content is one of the major bottlenecks of modern computer graphics. Good modeling tools must be intuitive, easy to use, robust and efficient. Most importantly, they should provide natural looking resultant models while requiring only minimal user interaction. Despite recent advances in geometry processing, modeling remains time consuming and requires both technical and artistic expertise. Hence, much of the emphasis in recent research is on simplifying the creation of new models.

One of the quickest ways to create a new model is simply by deforming an existing

one. Here, we propose an efficient algorithm for interactive model deformation in response to a simple user control mechanism in order to create a new model or animate an existing one. For example, given the user-prescribed positions for a set of control points on the surface of the model, the algorithm should compute the positions of the rest of the points in a manner that best preserves the original shape of the model. The method we propose is based on our novel local geometry representation, pyramid coordinates [70]. This representation captures the local shape properties of the model and is invariant under rigid transformations. Using this representation, we developed mesh editing operations that preserve the shape properties of the input models. As a result, our deformation method generates well-shaped models even under severe deformations. This dissertation also proposes an extension to pyramid coordinates called mean-value encoding [44]. Like pyramid coordinates, mean-value encoding is based on a set of angles and lengths describing the position of a vertex with respect to its neighbors. However, in contrast to pyramid coordinates, it uses a different local frame definition, which leads to a closed-form formulation. This enables us to achieve better results in terms of stability, speed, and shape preservation as compared to pyramid coordinates. Here, we also propose a new application, namely the realistic reconstruction of complex human motion based on motion capture (Mocap) data alone. We demonstrate that our algorithm constructs natural looking animated models and preserves the shape properties of the input models.

Another approach to simplify the creation of new models is to provide a simpler interface, for example a sketching tool [31]. Existing sketching tools are good for novices, but practical for creating only simple shapes. Our goal is to provide a tool with which almost anybody can create detailed geometric models quickly and easily. In particular, we propose a new technique for image-based modeling that allows a user to easily transform image contours into a 3D model with the help of a deformable 3D template. The image contours help inform the pose and proportions of the new shape, while the template model helps inform its full 3D shape and surface detail. The user input consists of tracing the image contours, as well as specifying a small number of point correspondences between the template model and the image. With this input, our proposed technique gradually deforms the template to fit the image contours. At

the heart of this process is the need to provide a good correspondence between points on image contours and vertices on the model. We propose the use of a hidden Markov model for efficiently computing an optimal set of correspondences. An iterative match-and-deform process then progressively deforms the 3D template to match the image contours. The technique can successfully deform the template model to match contours that represent significant changes in shape and pose. Our algorithm provides a powerful framework for developing detailed 3D models from images using only contour information and a flexible process for deforming 3D template models.

Many man-made and natural objects are easily classified into families of models with a similar part-based structure. Example families include quadrupeds, humans, chairs, and airplanes. This dissertation proposes a new method called Shuffler [41] which is a modeling system that automates the process of creating new models by composing interchangeable parts from different existing models within each family. Our system does not require the users to perform any geometric operations; they simply select which parts should come from which input model, and the system composes the parts together. To enable this modeling paradigm, Shuffler precomputes the interchangeable parts across each input family of models by first segmenting the models into meaningful components and then computing correspondences between them. We introduce two new algorithms to perform the segmentation and to establish intelligent part composition. By combining the two techniques we are able to identify and compose the interchangeable parts for a large variety of models. We provide a simple to use, and robust model composition system. Our approach significantly simplifies the generation of 3D content, making modeling faster, more accessible, and less expert driven.

Before we present our proposed algorithms for the cross-parameterization, model completion and model editing, we give an overview of the relevant previous work in the next chapter.

Chapter 2

Related Work

This section starts with a review of various cross-parameterization techniques used for morphing, shape blending, and transfer of texture or material properties. We then review various techniques for compatible remeshing required by morphing and blending applications. Next, we discuss several methods for model completion that are used to obtain a complete and consistent 3D model representation from incomplete range scan data. Finally, we review various techniques for novel 3D content creation through model deformation or the reuse of existing models, such as model composition or image-based modeling techniques.

2.1 Cross-parameterization

The ability to bijectively map one surface model to another is useful for many computer graphics applications. Recent research in digital geometry processing suggests multiple new applications for such a mapping, including pair-wise model editing [11], transferring texture and surface properties (BRDFs, normal maps, etc) [61], fitting template meshes to multiple data sets [7, 55]. Sheffer at al. [69] provide a review of cross-parameterization techniques developed for aforementioned applications.

Cross-parameterization typically needs to preserve the shape and features of the parameterized models, mapping legs to legs, ears to ears, and so on. The cross-parameterization is often computed by parameterizing the models on a common base domain. One popular choice for the base domain is the sphere. Number of algorithms exists for spherical parameterization, e.g. Alexa [3], Gotsman et al. [23], Praun and Hoppe [60]. Of those, only Alexa's method addresses feature correspondence. However, it does not guarantee a bijective mapping and is not always capable of matching

the features. An inherent limitation of a spherical parameterization is that it can only be applied to closed, genus zero surfaces.

A more general approach is to parameterize the models over a common base mesh [46, 52, 56, 61]. This approach splits the meshes into matching patches with an identical inter-patch connectivity. After the split, each set of matching patches is parameterized on a common convex planar domain. One advantage of this approach is that it naturally supports feature correspondence by using feature vertices as corners of the matching patches. The main challenge in mapping the models to a single base mesh is to construct identical inter-patch connectivities. The vast majority of the methods use heuristic techniques that work only when the models have nearly identical shape. Praun et al. [61] provide a robust method for partitioning both meshes into patches given user-supplied base mesh connectivity.

Template-based completion techniques, such as [6–8, 28, 34], compute cross-parameterization between the incomplete input mesh and the template to correctly close the gaps and holes. These techniques were tailored for reconstructing particular families of models. Most methods assume that the templates are geometrically very similar to the input meshes, in terms of shape [6, 34] or pose [7, 28].

Kraevoy et al. [45] introduce an algorithm that, given a mesh, a set of feature vertices, and a set of corresponding positions in the plane computes a patch layout and a triangulation of the points that have identical connectivity. Since the method operates in 2D, the authors were able to use standard parameterization techniques to achieve low distortion. Our work extends this general framework to partition meshes into patches with identical connectivity.

A concurrent work by Schreiner et at. [64] uses a similar procedure for base mesh construction, however, handling models of arbitrary genus more robustly. To generate a smooth cross-parameterization, they use a symmetric, stretch based relaxation procedure, which trades high computational complexity for quality of the mapping. To avoid artifacts, the method has to relax feature vertex correspondence in some cases.

2.2 Compatible Remeshing

Much of the previous work done in the area of cross-parameterization focuses on morphing and blending as target applications. Both morphing and blending require models to be represented by compatible meshes, i.e. meshes with identical connectivity. Alexa [2] gives a good review of cross-parameterization and compatible remeshing techniques developed for morphing.

Given the cross-parameterization, many techniques [3, 35] generate the common connectivity for the models by overlaying the meshes in the parameter domain and computing a common intersection mesh. The new mesh captures the geometry of the models. However, typically the new mesh is a factor of ten larger than the input meshes and contains poorly shaped triangles. Another alternative is to remesh the models using a regular subdivision connectivity derived from the base mesh [46, 56, 61]. Due to the rigid connectivity structure, the shape of the mesh triangles reflects the shape of the base mesh. Hence, if the shape of the triangles is poor, for example, if the user picked unevenly spaced feature vertices, the shape of the mesh triangles will reflect this. More importantly, a model that contains features interior to the base mesh triangles will require a very dense subdivision mesh over the entire model. Lin et al. [52] partly rectify this by introducing adaptive subdivision. However, their meshes still contain a very large number of elements.

Allen et al. [7] use the connectivity of one mesh to approximate the geometry of another, avoiding explicit parameterization. Their solution is limited to very specific inputs and can introduce severe approximation errors when the input models have significantly different geometry.

2.3 Model Completion

Modern commercial scanners, such as Cyberware [32], are becoming more robust and are capable of merging multiple scans using registration information. However, due to occlusions, the output meshes remain incomplete, containing numerous holes and multiple components. Several methods for closing holes were proposed recently [17,

48, 50, 59]. Davis et al. [17] successfully close holes in large meshes using volumetric techniques, but do not always preserve the topology of the mesh, generating spurious handles. Liepa [50] triangulates the holes using a method with $O(n^3)$ complexity, which can become unpractical for large meshes. Levy [48] uses 2D parameterization to efficiently close holes of any size. The technique requires manual alignment of the components in 2D to close gaps. Since the completion is performed in the plane, the method cannot generate closed models. Podolak and Rusinkiewicz [59] decompose the space into atomic volumes, and utilized the graph cuts to determine the individual volumes to be inside or outside. All four methods close the holes as smoothly as possible, without attempting to reconstruct the missing geometry. Sharf et al. [66], reproduce missing geometry by copying patches from other regions of the same model. However, they cannot generate missing features out of nowhere. Hence their methods are inadequate in the common case of scanned meshes missing complex features. None of the above methods uses global shape information. Therefore, they are unlikely to reconstruct correctly the connectivity for complex holes.

Template-based completion techniques, such as [6–8, 28, 34, 58], fill in the missing information using template geometry. These methods compute a cross-parameterization between the incomplete input mesh and the template to correctly close the gaps and holes. These techniques were tailored for reconstructing particular families of models. Most methods assume that the templates are geometrically very similar to the input meshes, in terms of shape [6, 34] or pose [7, 28] and the gaps and holes in the data are relatively small.

The method of Allen et al. [7] is one of the most robust. It uses user-specified correspondence between a small set of feature vertices, or markers, on the input and template meshes to align them with one another. The method has problems converging when the input models have significantly different geometry. The method of Pauly at al. [58] uses a database of 3D shapes to find a suitable template, it than warps the template to conform with the incomplete scanned data. The new method of Anguelov et al. [8] relies on the existence of an underlying skeleton to reconstruct models with major pose variations. As such the method is limited to human or animal models.

Sumner and Popovic [74] used a variation of the method in Allen et al. [7] to com-

pute mappings between models with significant shape variation. As they note, this often required specifying a very large number of marker correspondences.

2.4 Model Editing

Due to recent developments in digital geometry processing, mesh editing has been the subject of increasing attention in recent years. Geometry editing operations commonly use mesh encodings which capture the shape properties of the models. Given modified positions for a set of anchor vertices, the encoding is used to compute the positions for the rest of the mesh vertices, preserving the model shape as much as possible.

Much of the research in this area has focused on subdivision and multi-resolution editing techniques (e.g. Zorin et al. [82], Kobbelt et al. [40]). The idea is to decompose the mesh into two or more levels of detail, such that each level is encoded with respect to the previous one. The editing is performed on the coarsest level and then propagated to higher levels. Guskov et al. [25] develop an encoding where a vertex is encoded as a distance in the normal direction, from the average of the neighbor vertices. This provides a rotation-invariant representation of details. In order to use the encoding for editing purposes, most of these methods use a smooth base mesh as the coarsest level of the hierarchy. Thus, the general editing problem is reduced to the challenge of editing the smooth base mesh. Recent methods, such as Bischoff and Kobbelt [12], propose linear techniques for modifying the base mesh. However, since simple linear formulations are unable to distinguish between rotational and other linear transformations (Sorkine et al. [72]) the editing can lead to undesired artifacts.

A skeleton based encoding is an encoding where the position of each vertex is defined with respect to the links of the models skeleton. This encoding can facilitate simple editing operations. Although skeletons exist for any model in theory, they are generally hard to compute (e.g. Yoshizawa et al. [78]). Moreover, binary editing operations usually require skeletons with identical connectivity. The construction of such skeletons in 3D remains, to our knowledge, an open problem.

At the core of a surface-based mesh editing system lies a geometry encoding based on differential coordinates, such as Laplacian or gradient coordinates. Such systems

typically manipulate the differential coordinates of the undeformed mesh first, followed by a reconstruction of the deformed mesh from the modified differential coordinates. The Laplacian coordinates of each vertex are defined as a displacement vector between the average of the neighbor vertices and the actual 3D position of the vertex. Using this encoding, mesh editing becomes extremely efficient, since the decoding procedure only requires solving a simple linear system. Regrettably, since these coordinates are not invariant under rotation and scaling, the technique introduces visible artifacts for large deformations. Sorkine et al. [72] extend the use of Laplacian coordinates by linearly approximating local rotation and solving repeatedly for small rotational updates. Yu et al. [79] and Zhou et al. [81] combine Laplacian coordinates with a different rotation approximation mechanism. Zhou et al. [81], extends the Laplacian technique by constructing a volume graph at the interior of a closed mesh to prevent volume loss during excessive bending and twisting. Recently, Huang et al. [30] and Botsch et al. [14] have successfully introduced algorithms based on nonlinear formulations which also employ rotation estimation. Huang et al. [30] used a subspace domain to reduce the problem dimensionality via mean value coordinates [19, 33]. Botsch et al. [14] introduced a local shape representation based on prisms and used a hierarchial multigrid solver to reduce the problem complexity. Since all the above mentioned techniques use only an estimations of rotations the results are still suboptimal.

Lipman et al. [53] propose to split the editing problem into two separate linear systems. They first solve a linear system for per-vertex orientations, and from those reconstruct vertex positions in a second step. Since the first system does not consider position constraints, their technique neglects the connection between translations and rotations. While their method works very well even for large rotations, it exhibits shearing artifacts while performing large translations.

2.4.1 Modeling by Composition

Commercial modeling and design tools geared toward non-expert users typical utilize a procedural modeling approach where users can select shape components or properties from a restricted pre-processed dataset [51, 73]. Sketch based modeling interfaces [36]

allow users to create more diverse content, but are so far useful mostly for creating relatively simple shapes.

Mesh composition is emerging as an alternative simple modeling metaphor that allows even non-expert users to create complex models within a reasonable timeframe [20, 67]. In a typical composition setup, users first cut the two composed components from their respective input models and then align them such that the corresponding cut boundaries match reasonably well. The composition software is then used to connect the two components into a single model and define the geometry in the boundary region using a blending mechanism.

Several researchers proposed ways to simplify the user interaction during the boundary cutting and alignment [20, 26, 47, 67]. Smart scissoring methods [20, 47, 67] simplify the specification of the cut boundaries. Sharf et al. [67] require the user to provide only a coarse alignment and then improve the alignment further using a variation of deformable ICP. Funkhouser et al. [20] use the correspondence between the shuffled-in and out components to compute the alignment. This approach can lead to unintuitive alignment if the shuffled-in and out components differ significantly.

Our proposed modeling system combines new algorithms for meaningful mesh segmentation, part matching and intelligent part composition. This dissertation concentrates on the algorithms for meaningful mesh segmentation and intelligent part composition. Part matching algorithm was developed in a joint work with D. Julius and is not a part of this thesis.

We now review relevant existing methods in these categories.

Meaningful Segmentation: There is a large number of methods for meaningful segmentation of individual meshes, surveyed by Shamir [65] and Attene et al. [9]. Most methods rely on definitions of meaningful parts based on a study showing that humans segment models in regions of high negative curvature [29]. Geodesic distances are commonly used to steer the segmentation and negative curvature is taken into account when generating the actual cuts [37, 38, 80]. Amato et al. [49] consider the depth of the concave, negative curvature, regions on the model to determine where to generate the cuts between the components. Chaselle et al. [15] use a dual approach, where instead of generating cuts in concave regions they generate parts which correspond to

convex regions. The method searches for exact convex decomposition and shows that this problem is NP-hard.

Part Composition: Most composition tools focus on correct ways to combine the geometries of the parts in order to generate a smooth transition between them [72, 79], and rely on the user to specify the part boundaries on the input models and to align the parts with respect to one another. Recent approaches for simplifying the boundary specification process include intelligent scissoring [20, 47], which optimizes approximate boundaries generated by the user, and boundary computation based on part alignment [26, 67].

2.5 Sketch-based Modeling

Most commercial modeling systems, such as Maya [5] and 3D Studio Max [1], are targeted toward expert users and require significant time, expertise and artistic talent to generate complex models. Sketch based modeling interfaces [31] are much easier to manipulate and are successfully used to create new geometric models. Sketch-based modeling work has a strong connection with both image-based modeling and model-based vision, given that the types of contours that are useful to sketch are also often the same ones that are useful when attempting to infer geometric models from single images. The use of user drawing coupled with strong a priori knowledge about object classes has been used in [62] for modeling plants from a single photograph and in [76] for modeling airplanes, cups, and fish from sketched or traced contour lines. Earlier work [18] supports fast architectural reconstruction from images using a priori knowledge of typical shapes and user-driven selection of key image points. Model-based vision has a long history in computer vision. A prime example is [54], which uses matches between the silhouette edges of the known 3D model and edges extracted from the image in order to determine the 3D pose of the model.

Recently, two new papers introduce sketch-based deformation interfaces [39, 57]. The method of [39] uses as input a reference stroke on the model and a target stroke, indicating the deformed shape of the reference. Nealen et al. [57] infer the reference stroke from the target stroke, using a set of assumptions. The method computes sil-

houette edges or a suggestive silhouette within a user specified region and uses the silhouette as the reference stroke. User assistance is required if the silhouette needs to be trimmed or if more then one silhouette exists inside the region. Both methods assume arc-length correspondences between the points within each stroke, e.g., the half-way point of the reference stroke is chosen to correspond to the half-way point on the target stroke.

A key step of our proposed work is the automatic computation of optimal correspondences between points on the 2D target contours and the vertices on the 3D model that form its silhouette. It is related to the problem of computing optimal matches between 2D open curves. Hidden Markov models (HMMs) have been proposed as a technique for 2D shape classification using silhouette edges [10, 22, 27]. Typically, these models assume the existence of a significant number of training examples in order to learn the HMM parameters, and are commonly demonstrated working on closed silhouette curves only.

Chapter 3

Cross-Parameterization and Compatible Remeshing of 3D Models

The ability to bijectively map one surface model to another is useful for many applications. Smooth morphing of one geometric shape into another is one of the oldest and most popular special effects in movies. The first stage of most morphing algorithms is the establishment of a bijective mapping between the models [3, 35, 46]. Recent research in digital geometry processing suggests multiple applications for such a mapping, including pair-wise model editing [11], transferring texture and surface properties (BRDFs, normal maps, etc) [61], fitting template meshes to multiple data sets [7, 55], and morphing of one geometry into another [3, 35, 46]. Many of these applications, such as blending and morphing, require compatible input meshes, i.e. meshes with identical connectivity. Therefore, given the cross-parameterization, the models must be remeshed with a common connectivity.

The models which need to be cross-parameterized usually have similar features (there is little use for mapping a phone onto a cow) and the parameterization must respect those. For example, when mapping between two humans, the legs must map to the legs, the ears to the ears, and so on. This is typically achieved by enforcing the correspondence for a small set of feature vertices and using a cross-parameterization that preserves the shape of the models as much as possible (in terms of angles and area).

3.1 Algorithm

The input to the algorithm consists of two closed manifold meshes M_s and M_t and the corresponding sets of matching feature vertices V_s and V_t, typically selected by the user. The algorithm has three main stages. First, we construct a common base domain. Second, a low distortion cross-parameterization is computed. The final stage remeshes the input models in the mutually compatible way using the parameterization.

3.1.1 Common base domain construction

The goal of this stage is to construct a common base mesh domain for M_s and M_t based on the matching feature vertices V_s and V_t. To construct it, we compute topologically identical triangular layouts of the two meshes. The layouts are constructed incrementally by adding pairs of matching paths between feature vertices. Edge paths match if both their start and end vertices correspond to one another. When adding each pair of matching paths, the following conditions have to be satisfied:

1. **Intersection:** The new paths must not intersect existing paths in either of the mesh layouts P_s and P_t.

2. **Cyclical Order:** The paths must have the same cyclical order around the end vertices.

3. **Blocking:** The new paths must not block necessary future paths. Blocking happens when a path splits a patch in two and the corresponding feature vertices are placed in topologically different patches. Blocking is tested by propagating fronts from the same side of both paths (as determined by mesh orientation) and comparing the feature vertices encountered by the front.

The common base domain is constructed by incrementally introducing pairs of matching paths between marker vertices, pair-by-pair. At each point, the algorithm selects the pair of paths with the smallest length sum that satisfies the conditions described above. Cases exists in which its impossible to trace an edge path between feature vertices due to the mesh connectivity restrictions. Similar to Kraevoy et al. [45],

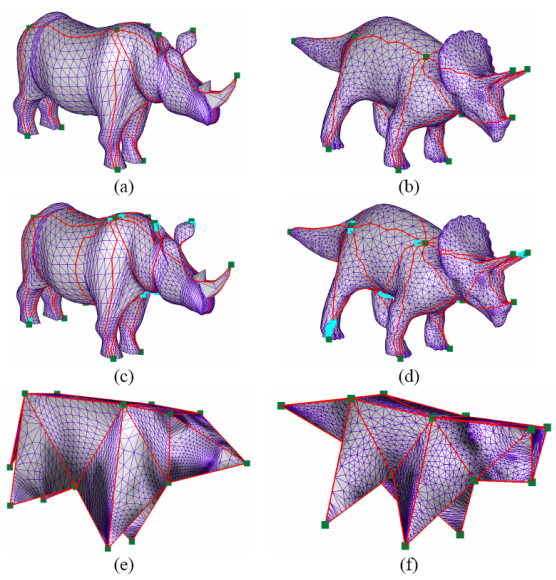


Figure 3.1: Base domains construction (feature vertices are dark green): (a),(b) simple paths; (c),(d) paths with additional vertices, new vertices are highlighted (turquoise); (e),(f) base meshes.

our algorithm adds extra, Steiner, vertices as necessary to resolve this problem (Figure 3.1 (c),(d)). The construction method is guaranteed to terminate and to compute topologically identical triangular layouts. The computed layouts are used to generate the cross-parameterization as explained in the next section.

3.1.2 Cross-parameterization

After the base mesh domain is constructed, each patch is mapped to the corresponding base mesh triangle using mean value parameterization [19]. This provides two parameterizations F_s and F_t between the meshes M_s and M_t and their corresponding base domains B_s and B_t (Figure 3.1 (e),(f)). By mapping each triangle in B_s to the matching triangle in B_t we obtain the initial cross-parameterization F. Since each component of the mapping is bijective, the combined parameterization is bijective as well. Mean value parameterization computes a shape preserving (conformal) parameterization. However if the patches are not well shaped the resulting mapping can still

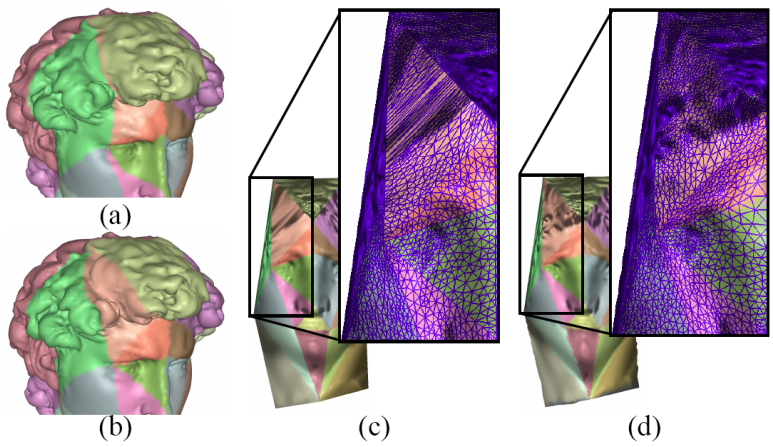


Figure 3.2: Parameterization on a base mesh: zoom-in on patches (a) and mapping to base mesh (c) before and after smoothing (b) and (d).

be quite distorted (Figure 3.2 (c)). To reduce the parameterization distortion we apply a smoothing procedure.

Our smoothing algorithm reduces the parametric distortion by allowing vertex migration from one patch to another, improving the patch shape (Figure 3.2 (a), (b)). To guarantee a bijective mapping, the smoothing procedure requires an adjacency constraint to be satisfied for all edges in the mesh. An edge satisfies the adjacency constraint if its end vertices belong to the same patch or to two adjacent patches that share a common boundary path. After the initial cross-parameterization the paths are aligned with the edges of the base mesh (Figure 3.2 (c)). Hence, the only edges that violate the adjacency constraint at this stage are edges whose vertices lie on two bounding paths of a single patch. Prior to running the smoothing procedure, each such edge is split into two. During smoothing, vertices are prevented from migrating from one patch to another if one of their edges will violate the adjacency constraint (Figure 3.3).

The smoothing is performed independently for M_s and M_t, modifying the current mapping from the mesh to the base domain. The smoothing procedure iterates over the mesh vertices, repeatedly modifying their locations on the base mesh. Given a vertex v located at the base triangle b, we map b and the three adjacent triangles ($b1$, $b2$ and $b3$) to a planar equilateral triangle E (Figure 3.3). We also map each neighbor of vertex v to

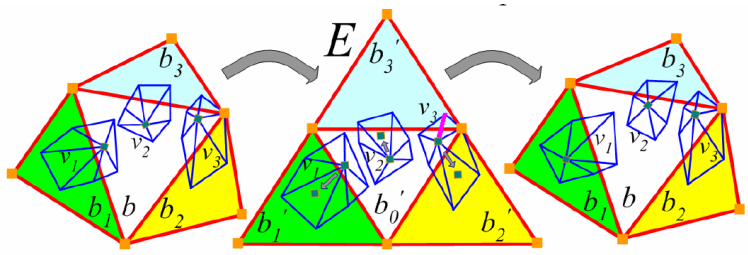


Figure 3.3: Relocating vertices on base mesh (v_3 violates the adjacency constraint).

E. Due to the adjacency constraint, each neighbor of v is located on one of the four base triangles b, b1, b2 and b3. Hence, this mapping is well defined. Given the locations of the neighbor vertices, we compute a new location v_e based on the locations of the neighboring vertices using the mean value parameterization Floater [19]. If moving the vertex to the new location v_e flips any triangles, we restore vertex to its original position v. The smoothing is repeated until the vertices no longer move or until a fixed number of iterations is reached. The procedure redistributes the vertices between patches, straightening the patch boundaries and significantly reducing the parametric distortion.

The parameterization framework introduced here is used in the following section to remesh the models in a compatible fashion.

3.1.3 Compatible remeshing

The remeshing algorithm constructs compatible meshes for the two models. The algorithm first remeshes the target model with the connectivity of the source mesh. The algorithm uses the cross-parameterization to map the vertices of the source mesh M_s to the target model M_t, creating a mesh M_{st} with the same connectivity as M_s. This yields compatible meshes for both models. However the mesh M_{st} may be a very poor approximation of the target geometry (Figure 3.4 (b)) in curved regions. The algorithm than improves the geometry approximation by vertex relocation and refinement. Since we want to minimize the final vertex count, refinement is performed only when strictly necessary.

The stages of the remeshing algorithm are as follows:

1. Compute the distance approximation error across the surface.
2. While error is above a given threshold:
   (a) Relocate the vertices of M_{st} on the target model using a smoothing procedure. Recompute the distance approximation error.
   (b) Refine the meshes where necessary using edge splits.

Error Computation

At each stage of the remeshing we need to calculate the approximation error between the target model M_t and its approximation M_{st}. Such error is typically computed using a discrete Hausdorff metric, which measures the distance from the vertices of one surface to the other surface. In our case, the vertices of M_{st} lie on the surface of M_t so the one-sided error is zero. We therefore need to measure only the distance between the vertices of the target mesh M_t and the approximation surface.

We define the mapping F' between M_s and M_{st} by mapping each vertex of M_s to the corresponding vertex of M_{st}. The distance error $e(v)$ at each vertex v of mesh M_t is defined as:

$$e(v) = (F'F^{-1}(v) - v)$$

The error inside each triangle of M_t is computed by interpolating the error of each vertex using barycentric weights. For any point p on the new mesh M_{st}, we map p to the target mesh using $F(F')^{-1}$ and use the error at that location. The face coloring in Figure 3.4 shows the approximation error throughout the remeshing procedure, with red denoting areas of higher error.

Adaptive Smoothing

The adaptive smoothing procedure modifies the parameterization of M_s on the base mesh B_s. This modifies the mapping F and as a consequence moves the vertices of the projected mesh M_{st} along the target mesh M_t. We apply the same smoothing procedure as in Section 3.1.2. The only difference between the two procedures is the choice of

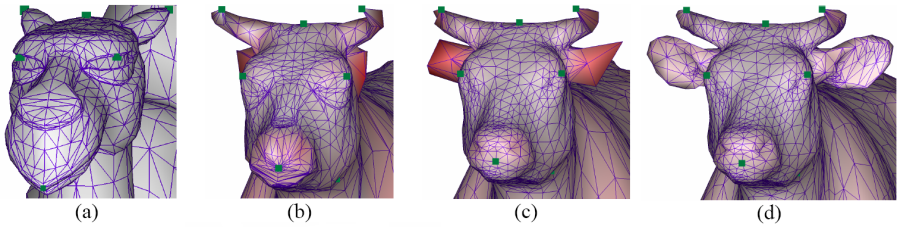


Figure 3.4: Compatible remeshing: (a) source mesh. (b) M_{st} after initial projection. (c) M_{st} after smoothing. (d) final mesh (after smoothing and refinement). There are no features on the camel corresponding to the cow's ears (mapping the camel's ears to the cow's horns provides a more natural morphing sequence).

weights. The adaptive smoothing algorithm assigns weights w_{uv} to the edges (u,v) using the following formula.

$$e_{uv} = (e(u) + e(v))/2$$

$$w_{uv} = (e_{uv} + w_{uv})/2$$

The edge error e_{uv} is based on a combination of the errors at the end vertices. The weight is the combination of the current error with the previous weight. This choice of weights attracts vertices to areas of higher error. The averaging with previous value of w_{uv} is used to avoid high weight fluctuations. The result is gradual mesh smoothing which prevents vertices from jumping back and forth when the error increases in one location and decreases in another. After each smoothing iteration, the positions of the vertices of M_{st} are recomputed.

Refinement

Adaptive smoothing goes a long way towards accurately approximating the target mesh with the projected mesh M_{st}. Clearly, however, if the target contains complex features not available in the source model (e.g. the cow's ears in Figure 3.4), they cannot be captured by the source connectivity. Therefore, when smoothing alone can not approximate the geometry accurately, the local mesh needs to be refined. This is done by performing edge split operations. The refinement uses the same error function e_{uv} as

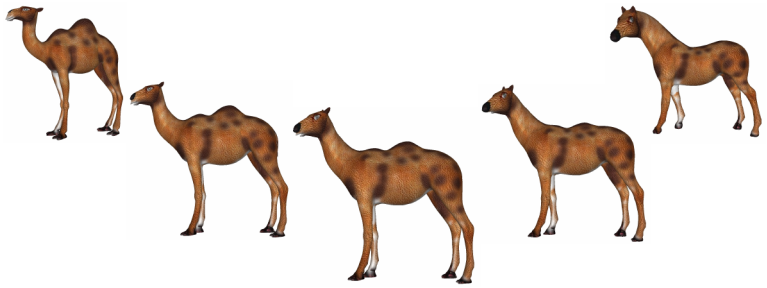


Figure 3.5: Texture transfer and morphing;

the smoothing. An edge is refined if

$$e_{uv} > \max(\frac{1}{2}\max_{(u',v')}(e_{u'v'}), \varepsilon)$$

where ε is the user-defined approximation error threshold. We found that refining only the edges with error larger than half of the current maximal error provides sufficient approximation accuracy without adding too many vertices to the final mesh. When the computed error exceeds the current threshold, the edge is split by placing a vertex at its mid-point. The split is performed simultaneously on M_{st} and M_s, thus preserving the mesh compatibility. To avoid unnecessary refinement edge splitting is performed only when smoothing alone no longer decreases the global error.

The result of the presented algorithm is a shape-preserving cross-parameterization between two compatible meshes Ms and M_{st} where M_{st} closely approximates the geometry of the target mesh M_t. The cross-parameterization maps the feature vertices of one model to the corresponding ones on the other and is guaranteed to be bijective.

3.2 Experimental Results

Figures 3.5 and 3.6 demonstrate the applications of the algorithm. Figure 3.5 demonstrates texture transfer performed using the cross-parameterization. By texturing the compatible meshes the texture can be used throughout a morphing sequence.

Figures 3.6 and 3.5 showcase the use of the computed compatible meshes for blending and morphing. The three-sided blends of a cow, rhinoceros and triceratops highlight

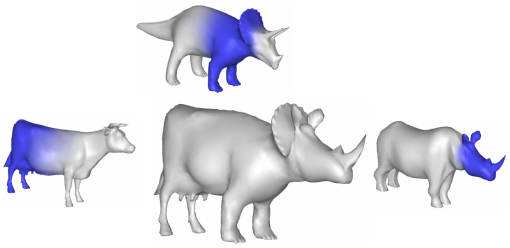


Figure 3.6: Three-sided blending.

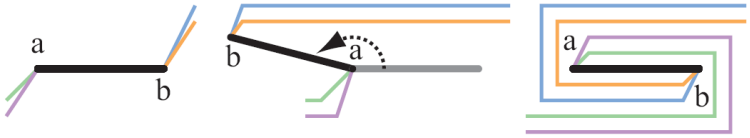


Figure 3.7: An example of a swirl: *b* turns around *a* pushing patch boundaries ahead of itself until *b* comes back to its original position (Praun et al. [61]).

the methods ability to simultaneously cross-parameterize and remesh multiple models. The algorithm takes about two minutes to run on models of up to 10K triangles on a 3GHz Pentium IV.

3.3 Conclusions

In this chapter, we have introduced new methods for computing shape preserving cross-parameterizations and compatible remeshing. The parameterization method is guaranteed to find a bijective, feature-preserving parameterization for any set of models. The smoothing mechanism we propose significantly reduces the mapping distortion compared to other parameterization methods. Our remeshing method generates compatible meshes for both models with fewer vertices than previous methods while accurately approximating the input geometry.

Limitations and Future Work

Compared to previous techniques, this method is significantly less dependent on the shape of the patches. However, visible artifacts can still occur when patch vertices have very high valence. The user can avoid these artifacts by specifying additional feature vertices.

In this work, we primarily focus on genus zero models. Although our technique carries over to higher-genus models it is not guaranteed to find a solution. Another problem may arise when paths between feature vertices take unnecessarily long routes around other existing paths. This *swirling* or *winding* phenomenon (Figure 3.7) leads to particularly badly shaped patches that cannot be fixed using local continuous relaxation [61]. Schreiner et al. [64] introduced a set of heuristics to change the order of path introduction to avoid the creation of swirls and to handle higher-genus models. To generate a smooth cross-parameterization, they use a symmetric, stretch based relaxation procedure, which trades high computational complexity for quality of the mapping. To avoid artifacts, their method has to relax feature vertex correspondences. An interesting topic for future research would be to explore different methods to avoid the creation of swirls completely and to handle models of arbitrary genus.

Chapter 4

Template-Based Mesh Completion

In this chapter, we extend our cross-parameterization technique to support models with multiple gaps and holes. This allows us to develop a new and robust method for template-based range scan data reconstruction.

Meshes generated by range scanners and other acquisition tools are often incomplete and typically contain multiple connected components with irregular boundaries and complex holes. We propose a robust algorithm for completion of such meshes using a cross-parameterization between the incomplete mesh and a template model. We employ this cross-parameterization to correctly glue together the components of the input mesh and to close the holes. The template is used to fill in the topological and geometric information missing in the input. The completed models are guaranteed to have the same topology as the template.

4.1 Algorithm

Our proposed algorithm operates on two meshes: input and template. Like previous template-based techniques, we use a small set of markers to provide a coarse alignment between features on the input and template meshes (Figure 4.3 (a) and (b)). The markers are specified either manually or provided as part of the data [7].

Our algorithm has four main stages, visualized in Figure 4.3:

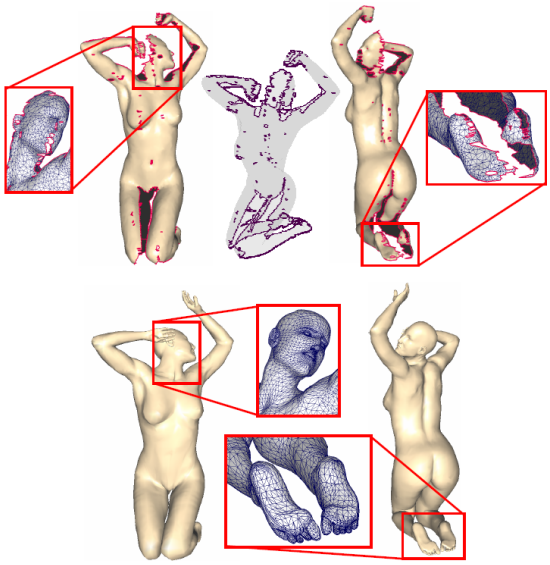


Figure 4.1: Template-based mesh completion: (top) incomplete scan input the semi-transparent grey image shows the multiple complex holes in the input; (bottom) reconstructed model.

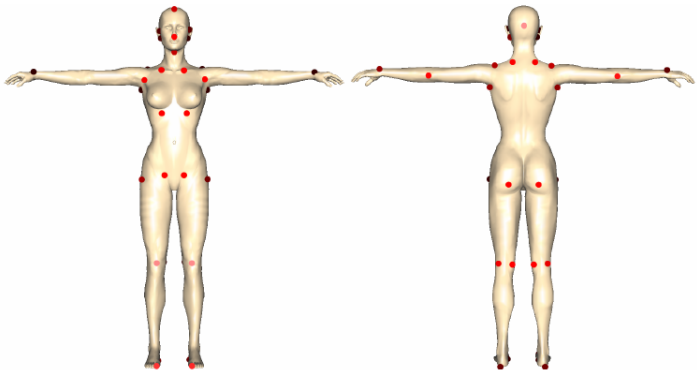


Figure 4.2: Template and markers used to complete the female model in Figure 4.1.

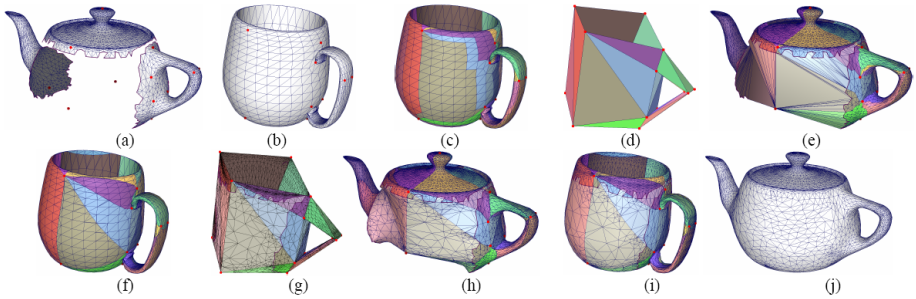


Figure 4.3: Algorithm stages. (a,b) Input and template meshes with markers (c-e) Segmentation: (c) template, (d) base mesh, (e) input segmentation and virtual triangulation (virtual triangles shown in lighter color). (f-i) Parameterization: (f) template; (g) input mapped to base; (h) closed input mesh; (i) input mesh mapped to template; (j) completed model. Note that the shape of the sides and the bottom comes from the mug and hence differs from that of the traditional teapot.

4.1.1 Pre-processing

In this stage the algorithm prepares the input mesh for cross-parameterization by editing mesh components with multiple boundary loops. The algorithm unites the boundary loops into one. Using the face graph of the mesh, it first computes a shortest path tree connecting the loops. Then it merges the loops by removing the faces in the tree from the mesh. The purpose of pre-processing is to avoid ambiguities in gap closure, as it is often difficult to classify which boundary loop on one mesh component corresponds to a boundary loop on another. Generating a single boundary for each component eliminates this ambiguity.

Next, the algorithm segments the pre-processed meshes into patches.

4.1.2 Segmentation and base-mesh construction

A consistent mesh segmentation for closed meshes is typically constructed by incrementally introducing paths between marker vertices [42, 61, 64]. First, similar to Kraevoy and Sheffer [42], the template mesh is segmented by introducing legal edge paths between marker vertices, one-by-one (Figure 4.3 (c)). Given the segmentation, the algorithm constructs the corresponding base-mesh by generating straight edges be-

tween the corner vertices of the patches (Figure 4.3 (d)).

To generate the input mesh segmentation consistent with the template mesh segmentation, the algorithm introduces paths between marker vertices one-by-one, based on the base-mesh connectivity. When introducing each path we must make sure that it does not block future paths [61]. Regrettably, in a multi-component setting path blocking cannot be tested as described in Kraevoy and Sheffer [42]. Therefore we specify a particular order of adding paths in which blocking cannot occur. Our method introduces paths that correspond to base mesh edges in a specified order.

First, for each connected component, the method constructs a Steiner tree of the markers in this component using only interior paths. Next, it proceeds to generate a Steiner tree of the connected components, by adding cross-gap paths between markers in different components. At the end of this stage, all the markers are connected into a single tree, while the paths constructed so far do not block any future paths between markers inside any component or between markers in different components. At the same time, from now on, new paths will not cause blocking, as all the markers are connected. Finally, the algorithm completes the segmentation by introducing the rest of the paths, each time adding the shortest path that corresponds to a base-mesh edge.

The input and template meshes are now parameterized onto the base mesh, using the computed segmentation.

4.1.3 Base-mesh parameterization

This section presents a framework for mapping meshes with multiple components and holes onto a base mesh. The framework enforces the mapping of markers on the input and template meshes to the vertices of the base mesh. To correctly parameterize gaps and holes we use virtual triangulation. In the past, virtual triangulation was used as pre-processing for planar parameterization, to enable parameterization of meshes with holes [68] and to reduce parameterization distortion. Our work further develops this idea using iterative Delaunay triangulation to improve the quality of parameterization and facilitate automatic gap and hole closure.

The mapping is computed in two steps. An initial one-to-one embedding from the

mesh to the base domain is computed by mapping each patch onto the corresponding base-mesh face. The method applies uniform embedding [75] to parameterize each patch onto the base triangle. Note that a region of a gap or a hole, that is associated with each patch, is by construction a simple polygon, hence, the mapping is guaranteed to be bijective. Next, the algorithm uses the mapping to triangulate the gaps and holes in the patch, enforcing base-mesh topology. The mapping and the virtual triangulation are then improved using a combined smoothing and re-triangulation procedure (Figure 3 (f-i)). It iterates over all the pairs of adjacent faces in the base mesh, applying the following steps:

1. **Parameterization:** For each pair of adjacent base-mesh triangles, Step 1 constructs the quadrilateral domain and parameterizes the corresponding mesh patches in this domain, using the parameterization method proposed by Yoshizawa et al. [77].

2. **Remeshing and projection to 3D:** This step re-triangulates the holes and gaps contained in the two parameterized patches using Delaunay triangulation. Extra vertices are added as necessary to improve the triangulation quality [71]. To compute the 3D positions for the new virtual vertices, we use mean value embedding [19] in 3D with the weights computed in the plane (Figure 4.3 (h)). The re-triangulation improves both the parameterization and the gap and hole closure.

The smoothing and triangulation procedure is repeated until both the parameterization and the triangulation no longer change. By repeatedly smoothing pairs of mesh patches, we allow vertices to move freely all across the base. Figure 4.3 (g, h) shows the result of applying our algorithm to parameterize and triangulate the broken teapot model. The parameterization preserves the shape of the input mesh and in particular the outline of the boundaries. The gap closure is close to optimal, with the correct components connected to each other.

4.1.4 Blending

For template-based completion we use the mapping computed by the previous stage to construct a complete geometric model (Figure 4.3 (j)) by blending the input and template meshes.

To facilitate blending we establish common connectivity for the input meshes. We use the connectivity of the closed input mesh (Figure 4.3) or the template (Figure 4.1) as the basis for common connectivity. The choice depends on the complexity and resolution of the two meshes. Using the finer, more detailed mesh provides a better approximation. Similar to Kraevoy and Sheffer [42] the algorithm refines the connectivity if it fails to capture all the details on one of the models. Since the input and template models often have very different shape, we cannot blend the 3D coordinates directly. To provide intuitive results we blend local shape descriptors instead. In our examples we used the blending scheme of Sheffer and Kraevoy [70] to generate the completed models.

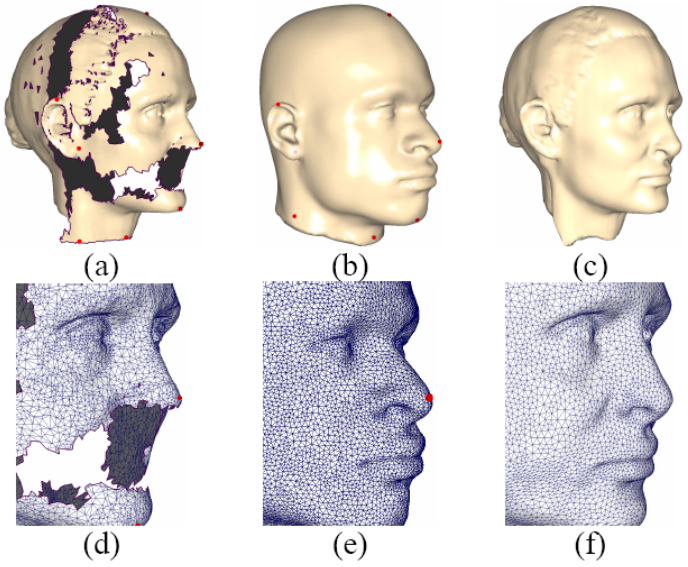


Figure 4.4: Completing a head from 3 fragments: (a,d) the input mesh, (b,e) the template, (d,f) the completed model. Features, such as nostrils and mouth, not available in the input mesh were taken from the template.

4.2 Experimental Results

Here, we demonstrate several models constructed using our scheme. Global completion took from 20 seconds to 8 minutes for models of 10K to 200K triangles. The times were measured on a 3GHz P4. These times are comparable with those we attained from the authors of [7]. However, our method is significantly more robust, succeeding on models, such as Figure 4.1.Our parameterization algorithm is significantly faster than previous cross-parameterization techniques [42, 64].

Figure 4.1 demonstrates the results of using our method for completion of large and complex holes. The scanned female model in Figure 4.1 differs significantly in terms of pose and body proportions from the canonical template that we used (Figure 4.2). Despite these differences the completion algorithm correctly reconstructs the cross-hole connectivity and accurately completes the missing features based on the template.

Figure 4.4 shows the reconstruction of a head from several components. Thanks to the near-perfect alignment achieved by our mapping, the method seamlessly blends partial features from the template and input meshes. In this example the upper half of the nose comes from the input while the lower half comes from the template, yet the two are blended seamlessly and the original geometry of both is clearly preserved.

There are many models for which no standard template exists. Such is the case for the teddy bear model generated from merged Z-camera scans (Figure 4.5). Nevertheless, the model is successfully completed using a simple sphere as a template. The markers are placed automatically at four roughly equidistant (in Euclidean space) points on the model and the sphere. The method robustly parameterizes and completes the model from eleven connected components, most of which have no markers. The spherical parameterization (Figure 4.5 (b)) is globally continuous and has low distortion.

4.3 Conclusions

We proposed a robust new method for template-based mesh completion. As demonstrated by the examples, our new method is robust in the face of incomplete input

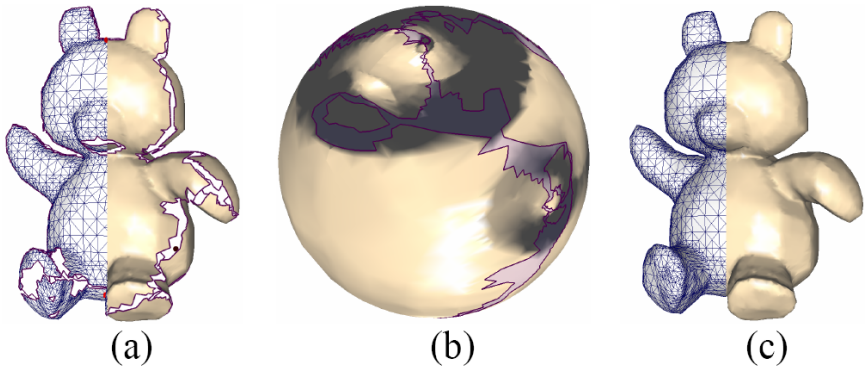


Figure 4.5: Completing teddy from 11 components using the sphere as a template and 4 markers: (a) input, (b) teddy mapped to the sphere (normal map), (c) completed model.

meshes with multiple components and holes. It can successfully complete models on which previous methods are likely to fail. It is very efficient and requires only a small amount of user interaction to specify the marker vertices. Our parameterization mechanism is more generic than previous techniques; it guarantees bijectivity and introduces less distortion.

Limitations and Future Work

Due to noise, self-occlusions and collisions in the scanned data, the input mesh can have a different genus compared to that of the template mesh provided by the user. Since bijective mapping between surfaces of different genus is mathematically impossible, differences in genus will lead to algorithm failure. An interesting topic for future research would be to explore automatic ways of detecting genus inconsistences in the input data based on the user provided template model and a set of markers specifying the correspondence between the two.

The scan data can also be extremely noisy due to scanning system errors.Our blending procedure uses information from the template in areas where the input geometry is missing or unreliable, and the input geometry where it is available, thus carrying the noise over to the final model. It is possible to reduce the amount of noise by marking the noisy regions as unreliable. This will result in partial blending between the noisy

regions and the template reducing the amount of noise. Automatically detecting the locations of noisy regions and the amount of blending required might also be a new interesting research topic for investigation.

Chapter 5

Pyramid Coordinates for Morphing and Deformation

In previous chapters we introduced new methods for computing shape preserving cross-parameterizations and compatible remeshing, which are suitable for novel 3D content creation through reuse of existing models by transferring surface properties from one model to another.

Another means by which we can create novel 3D content is through geometry manipulation. Mesh editing operations such as deformation, morphing and blending are useful extensions to our modeling pipeline, and are powerful editing tools that lie at the heart of many geometry processing applications in computer graphics and geometric modeling. Good editing tools must be intuitive, easy to use, robust and efficient. Most importantly, they should provide natural looking resultant models while requiring only minimal user interaction. While it is somewhat difficult to quantify the intuitiveness or natural look of edited models, the consensus seems to be that the models should preserve as much as possible their local and global shape properties (distances, angles, etc.) [2].

This chapter focuses on two types of editing operations:

Deformation: Mesh deformation is the process of interactively transforming the surface of a model in response to some control mechanism. It is commonly used for model editing and animation. Typically, mesh deformation techniques require global knowledge of the model structure (such as a skeleton) which are quite time consuming to compute. We propose a new approach for 3D mesh deformation based on a small number of user-specified *control vertices*. Given the positions of the control vertices,

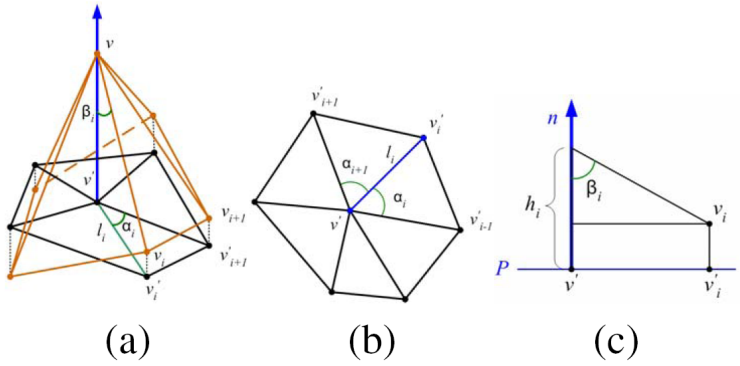


Figure 5.1: (a) Pyramid coordinates: (b) tangential components in the projection plane P; (c) normal component β.

our method computes the positions of the rest of the vertices, in a manner that best preserves the shape parameters of the source model. As demonstrated by Figure 5.2, we generate natural looking deformations in seconds with minimal user interaction.

Morphing: Morphing is one of the basic and most popular computer graphics applications. Morphing algorithms create a smooth transition in time between multiple input models. The primary challenge of all morphing algorithms is to generate intermediate models that retain the appearance and properties of the source models. The proposed morphing procedure generates intermediate models which interpolate the shape properties of the input models. The algorithm also supports introduction of user defined trajectories for a number of control vertices.

5.1 Algorithm

The basic idea behind our algorithm is to describe the position of each vertex with respect to its neighbors in the mesh, rather than with respect to a global coordinate system. We would like the description to be invariant under rigid transformations in order to be able to move, bend or rotate parts of the model, such as limbs. Our representation, pyramid coordinates, is based on a set of angles and lengths relating a vertex to its immediate neighbors (Figure 5.1).

Pyramid coordinates: Let v be a mesh vertex in 3D and let $v_1, v_2, ..., v_m$ be its neighboring vertices. We define the projection plane

$$P = n_x x + n_y y + n_z z + d$$

using the normal at v, $n = (n_x, n_y, n_z)$, and $d = -\sum_{i=1}^{m} n \cdot v_i$. We define the projections of v and v_i to P as v' and v'_i respectively. The description of the vertex with respect to the neighbors consists of: a set of angles α_i between the projected edges $\langle v', v'_i \rangle$ and $\langle v', v'_{i+1} \rangle$; a set of angles β_i between n and the edges $\langle v, v_i \rangle$; and, a set of projected edge lengths $l_i = \|v' - v'_i\|$.

Reconstruction: Given those values we can uniquely define the vertex position given the neighbor vertices. We derive v' from v'_i using the reproduction property of the mean value coordinates [19]:

$$v' = \sum_{i=1}^{m} w_i v'_i, \tag{5.1}$$

where the barycentric weights w_i are derived from α_i and l_i. To derive the position of v given v', we calculate a set of offsets along n, $h_i = \|v' - v'_i\| \cot(\beta_i) + (v_i - v'_i) \cdot n$. Finally we obtain v by offsetting v' by the average of h_i along n as follows:

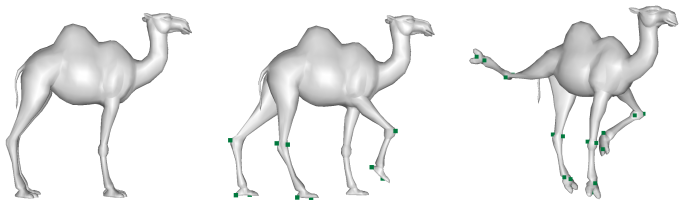
$$v = v' + n \frac{1}{m} \sum_{i=1}^{m} h_i. \tag{5.2}$$

The presented shape description is invariant under rigid transformations. Given a 3D model, the angles and lengths are uniquely defined, though more than one model can fit a given combination. We use this redundancy to facilitate the deformation process, generating models which closely preserve the shape of the input models, subject to deformation.

Deformation: Given user prescribed positions for a set of control vertices V_c, the positions of the rest of the vertices are computed by iterating the following scheme:

1. For each vertex v, if v is not in V_c, update the position of v using the shape description in Equations 5.1 and 5.2.

2. For each control vertex v in V_c, compute the height offsets h_i and update the positions of the neighbor vertices:



(a) (b) (c)

Figure 5.2: Examples of model editing; (a) source model (4884 faces); (b) walking camel - moderate deformation (10 control vertices, 3 sec); (c) dancing camel - extreme deformation (16 control vertices, 6 sec). Computations were done on P4 2.4 Ghz computer.

$$v_i = v_i' + n\frac{1}{m}\sum_{i=1}^{m} h_i.$$

This sets the distance between v and the projection plane P to the average of the h_i values.

3. Repeat until convergence.

The shape preservation metrics that we use focus on angles. However, under deformation, the angles are sometimes preserved at the expense of stretch. To account for stretch we use a simple strategy of scaling the edge weights w_i by $\frac{\|v'-v_i'\|}{l_i}$. In addition to better length preservation, the use of a stretch component during vertex placement drastically speeds up the convergence of the reconstruction procedure.

Morphing: Our method generates intermediate models based on interpolated pyramid coordinates. It can also take into account user-prescribed trajectories of a number of control vertices. The prescribed trajectory of a control vertex is a curve in space-time defined over [0,1].

Given source and target meshes S and T sharing a common connectivity, the algorithm computes the pyramid coordinates of S and T: projected face angles α^S and α^T , normal-edge angles β^S and β^T , and projected edge lengths l^S and l^T. For each time-frame, the algorithm computes the pyramid coordinates given the frames time t

(in [0,1]) by linearly interpolating between source and target values:

$$\alpha = \alpha^S(1-t) + \alpha^T t,$$

$$\alpha = \beta^S(1-t) + \beta^T t,$$

$$\alpha = l^S(1-t) + l^T t,$$

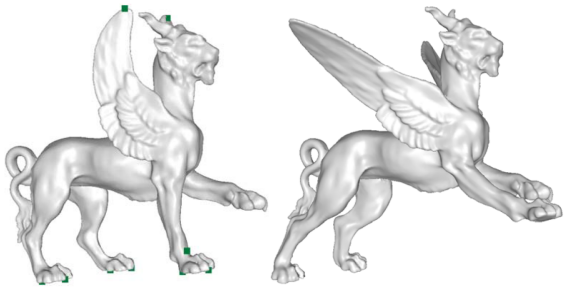
The control vertex positions are now set as defined by the trajectories. The remaining vertex positions are computed by the controlled reconstruction procedure using the pyramid coordinates. The initial guess for the placement is the vertex positions from the previous time-frame. Those positions are typically close to the expected positions in the current time step. Therefore, the reconstruction procedure converges very fast for each time-frame.

5.2 Experimental Results

The figures presented in this chapter demonstrate the results of applying our technique for different editing operations. Figures 5.2 and 5.3 showcase our deformation algorithm on the camel and feline models. Both models undergo large deformations using only a small number of control vertices. The resulting models closely preserve the shape of the original models. Figure 5.4 shows a morphing example where a cow is morphed into a bull. The morphing uses 8 trajectories to define the transition for the legs, tail, horns, and nose. Our morphing procedure generates intermediate models that retain the appearance and properties of the source models. Table 5.1 provides statistics and runtime results for the examples described above.

Model	Size	Control	Runtime
	#vert.	#anchors	(sec.)
Walking camel	4884	10	3
Dancing camel	4884	16	6
Feline	49,864	9	161
Cow/bull	21,610	8	-

Table 5.1: Deformation statistics.



(a) (b)

Figure 5.3: Deformation of feline model (a) source model (49,864 vertices) (b) jumping feline (9 control vertices).

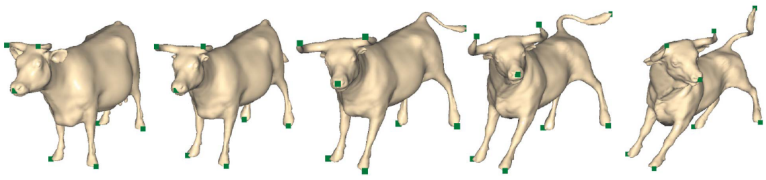


Figure 5.4: Turning a cow into a bull. Note the preservation of local details and the smooth rotation of the head and tail.

5.3 Conclusions

We introduce a new, robust method for mesh editing based on a local shape representation, which is invariant under rigid transformations. The sole input from the user consists of a number of control vertices defining the deformation. Given the input, our method provides natural deformations that include scaling, bending, and rotation of mesh parts, providing an effective tool for model editing, morphing and character animation.

Limitations and Future Work

Our method relies on tangential plane calculation for establishing a local coordinate frame. Computation of the tangential planes can be unstable on meshes with irregular connectivities. In the next chapter we propose an extension to pyramid coordinates called mean-value encoding. In contrast to pyramid, it uses a different local frame

definition, leading to a more stable closed form formulation. To improve performance even further, we introduce a multiresolution structure into the reconstruction procedure, interleaving it with the numerical minimization. This enables us to achieve much better results in terms of stability, speed, and shape preservation compared to pyramid coordinates.

Chapter 6

Mean-Value Geometry Encoding

In this chapter we propose an extension to pyramid coordinates called mean-value encoding. Like the pyramid coordinates, mean-value encoding is based on a set of angles and lengths describing the position of a vertex with respect to its neighbors. However, in contrast to pyramid, it uses a different local frame definition, leading to a closed form formulation. This enables us to achieve much better results in terms of stability, speed, and shape preservation compared to pyramid coordinates. Here, we also propose a new application namely the realistic reconstruction of the animated geometry of the human form based on motion capture (Mocap) marker information alone.

With the improvement and declining costs of motion capture technology, modern computer graphics increasingly uses it as a major source of data for character animation. Mocap data provides trajectories for character animation by tracing the motion of a set of markers on moving subjects. Standard techniques for motion data reconstruction from such data require global knowledge of the model structure, such as a skeleton. Skeletons are difficult and quite time consuming to construct. We propose the first, to our knowledge, automated technique that can realistically reconstruct character animation based on Mocap data alone (Figure 6.5(b)).

6.1 Algorithm

Given a mesh model and Mocap data, the sole input required from the user is a correspondence between markers in the Mocap data and vertices on the model (Fig-

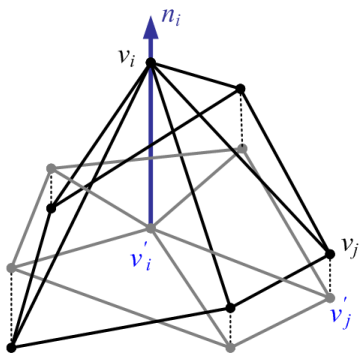


Figure 6.1: Mean-value encoding: The 3D mesh is shown in black, the normal n_i is shown as a vertical vector, the projected mesh in the local projection plane is shown in gray.

ure 6.5(a)). The correspondence provides the positions for a subset of the model's vertices for each frame in the animation sequence. Our goal is to find the positions for the remaining vertices in a manner that best preserves the shape of the source model. Model editing addresses a similar problem where the surface of the model is modified in response to some control mechanism, preserving the shape of the surface as much as possible. Recent methods for model editing, e.g. [79], require both the positions of the control vertices and their orientations, which are not available in the Mocap data. We introduce a new method for motion reconstruction using an editing approach that does not require orientation information. Given the positions defined by the Mocap data for a subset of vertices we use a new local shape representation to compute the positions for the remaining vertices. Our geometry representation stores the position of each vertex, with respect to its neighbors in the mesh, using a local projection plane, similar to [70].

Given a mesh with vertices V and edges E the projection plane corresponding to the mesh vertex $v_i \in V$ is defined in terms of a normal n_i and an offset d_i from the origin. We define a projection plane normal (Figure 6.1) as

$$n_i = \sum_{k=1}^{m} (v_{j_{k+1}} - l) \times (v_{j_k} - l) / \| \sum_{k=1}^{m} (v_{j_{k+1}} - l) \times (v_{j_k} - l) \| \tag{6.1}$$

where $v_{j_1}, \ldots, v_{j_m}$ are the neighbor vertices of v_i, and $l = \frac{1}{m} \sum_{(i,j) \in E} v_j$. In other words,

we use an area averaged normal to a local Laplacian mesh as the normal of the projection plane. This enables us to achieve much better results in terms of stability and shape preservation compared to [70], where the normal estimate was based on the current position of v_i. The local representation of each vertex with respect to its neighbors consists of: a set of mean-value coordinates w_{ij}, describing the vertex position in the projection plane relative to its neighbors; and, an offset h_i, describing the vertex offset above the projection plane. Unlike [70], we use a normal formulation based solely on the neighbor vertex positions v_j. Therefore, we are able to obtain an explicit formula for reconstructing v_i

$$v_i = F_i(V) = \sum_{(i,j)\in E} w_{ij}(v_j - (d_i + v_j \cdot n_i)n_i) + h_i n_i, \tag{6.2}$$

which leads to a closed form global reconstruction formulation

$$\arg\min_V G(V) = \frac{1}{2} \sum_{v_i \in V} (v_i - F_i(V))^2. \tag{6.3}$$

We solve this minimization problem using the Gauss-Newton method.

To enable near real time performance, we incorporate a multiresolution structure into the reconstruction procedure (Figure 6.2), interleaving it with the numerical minimization. The multi-resolution hierarchy is constructed using a simplification procedure that removes the non-marker vertices one by one, until only a base mesh connecting the markers remains. The simplification is performed using a sequence of half-edge collapse operations. A mesh hierarchy is constructed, keeping track of all the individual edge collapse operations. Before each collapsed vertex is removed, the mean-value encoding of the vertex in the current mesh is computed and stored for reconstruction purposes. The mesh is reconstructed by first placing the marker vertices at the specified locations (e.g., Figure 6.2 (c)). The subsequent reconstruction involves two major operations: vertex split and optimization.

Vertex split: Reversing the simplification order, collapsed vertices are added to the mesh one at a time. We use Equation 6.2 to obtain the position for each new vertex. Note that at the time of insertion, the positions of adjacent vertices in the current mesh are well defined. If the marker positions are unchanged or a rigid transformation of the original position, this placement gives the exact desired position of the vertex in 3D.

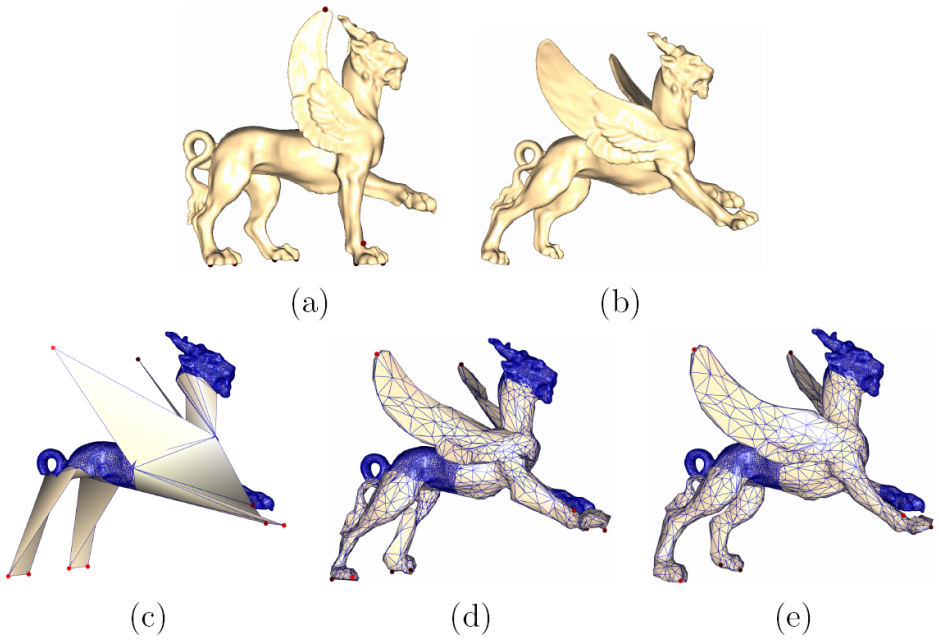


Figure 6.2: Deformation using mean-value encoding and decoding: (a) Original model. (b) Final mesh. (c) Base mesh (markers and vertices not affected by the deformation) with modified marker positions. (d) Intermediate mesh after several edgesplits. (e) Intermediate mesh after relaxation. Note the smoothing effect on the legs and the wings.

(a) (b) (c)

Figure 6.3: Comparison of deformations performed with and without multiresolution. (a) Original model. (b) Deformation with multiresolution. (c) Deformation without multiresolution.

Optimization: If the anchor positions are modified, each split introduces some error. While after each vertex split operation, $v_i - F_i(V)$ equals 0 at the inserted vertex v_i, this is not necessarily the case for the adjacent vertices. Hence $G(V)$ (Equation 6.3) is not optimized. To find the minimizer of $G(V)$, after performing a sequence of vertex splits, we use a Gauss-Newton minimization procedure combined with line-search. For the models we edited, we found it is sufficient to perform optimization only once during the reconstruction procedure for an intermediate mesh with about 3% of the vertices.

Figure 6.2 shows the reconstruction stages for a deformed feline model. The parts of the model that remain fixed, such as the head, are treated as markers. The error introduced by performing edge-splits alone is clearly visible on the intermediate mesh (Figure 6.2(d)). For this 100K triangle model the reconstruction took 0.86 seconds.

Figure 6.3 demonstrates the difference between the deformations performed with and without the multiresolution structure. The local details are nicely preserved in both cases. However, due to relaxation of the intermediate mesh, the global shape preservation is significantly better on the model deformed using multiresolution structure (Figure 6.3 (b)) . Not only is the global shape of the wings of the multiresolution example much closer to the shape of the original feline wings, but the deformation itself is evenly spread out over entire wing span. In contrast, the deformation without the multiresolution setup concentrates most of the change at the tips of the wings and as a result they become overstretched.

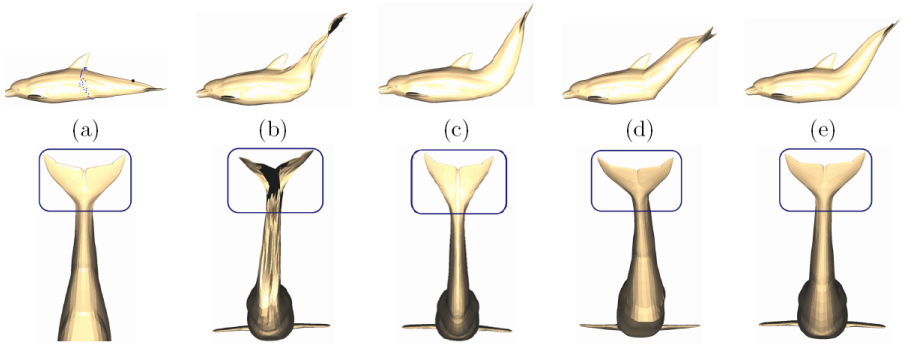


Figure 6.4: Comparison of deformation methods, with details (zoom in on the tail): (a) Original model. (b) Laplacian coordinates; (c) Extended Laplacian coordinates [72]; (d) Pyramid coordinates [70]; (e) Mean-Value encoding. Note that only the last example preserves the original shape of the tail fins.

The multiresolution approach makes the shape representation computation slightly more time consuming, but dramatically speeds up the reconstruction. The reconstruction takes less than a second for models of up to 100K faces. In case of Mocap data reconstruction the representation computation can be done once as a pre-processing step, while reconstruction is performed repeatedly and ideally should be done in real-time. Therefore, the hierarchical approach is very suitable for our scenario.

Figure 6.4 uses a simple example to compare our deformation technique to the deformations generated by several existing techniques. The deformation was performed using one control vertex at the tip of the dolphin's tail. The region of influenced used in the examples is marked by the blue dots (Figure 6.4 (a)). In all the examples the control vertices positions are identical and the regions of influence are the same. As expected, a purely linear deformation technique, such as Laplacian coordinates [4] (Figure 6.4 (b)), leads to extreme shear when the anchor position undergoes rotational displacement. The method of Sorkine et. al [72] (Figure 6.4 (c)) significantly reduces the distortion, but still leads to visible shearing artifacts near the tail fins. The Pyramid coordinates method, described in Chapter 5, (Figure 6.4 (d)) causes less shearing, but exhibits discontinuities near the anchor and along the boundaries of the region of influence. In contrast, our mean-value encoding and decoding mechanism produces a

smooth and intuitive deformation with no undesirable artifacts (Figure 6.4 (e)).

Approach	Translation	Small Rotation	Large Rotation	Complexity
Mean-Value Encoding	+	+	+	Non-linear
PriMo[14]	+	+	+	Non-linear
Botsch et al. [13]	+	+	−	Linear
Sorkine et al.[72]	+	+	−	Linear
Lipman et al.[53]	−	+	+	Linear

Table 6.1: General comparisons of deformation techniques under various types of deformations. Pluses indicate the techniques produce well defined intuitive deformations and minuses indicate that the technique fails and produces artifacts.

In table 6.1 we compare different mesh editing techniques in terms of general performance under various types of deformations. Note that our goal is to show under which circumstances each individual method fails.

The first two deformation techniques we examine are Mean-Value Encoding and the recently introduced PriMo [14]. Both are non-linear surface deformation techniques and, as a result, do not suffer from any linearization artifacts. However, non-linear techniques are computationally and implementation-wise more involved than their linear counterparts.

Another common approach for mesh deformation is to use physical simulation [21, 24]. The surface is assumed to behave like a physical skin, *a thin shell*, which stretches and bends as forces are acting on it. Mathematically, this behavior can be captured by an energy functional which penalizes stretch or bending. Physics based methods provide accurate model behavior, but they are often not intuitive to control and are usually relatively slow. Botsch et al. [13] introduced a new method where the optimal surface is the one that minimizes the energy functional while satisfying all the prescribed boundary conditions. The method work fine for pure translations (Table 6.1), i.e., it yields a smooth deformation and locally rotates the geometric details. However, due to linearization the method has problems with large rotations, such that the deformed surfaces exhibit loss of details specially in the regions with large protruding features.

Laplacian surface editing [72] implicitly determines the per-vertex rotations, and hence works similarly well for translations and small rotations. Its main drawback is

the required linearization of rotations, which yields artifacts for large local rotations.

Lipman et al. [53] solves a linear system to preserve the relative orientation of the local frames, which works well for any kind of rotations. However, since this linear system does not consider positional constraints, this method exhibits shearing artifacts under large translations.

The following guidelines for picking the 'correct' deformation can be derived from the above discussion:

In technical, CAD-like engineering applications required deformations are typically small, since the existing prototype only has to be adjusted slightly, but there are high requirements on surface fairness, boundary continuity and the precise control. For such kind of problems a linearized shell model [13] is a good choice. For applications like character animation that mostly involve large rotations of limbs, methods that are based on local coordinates are clearly the best. If the required rotations are available from, e.g., a sketch interface, the method of Lipman et al. [53] might be the better choice. In cases when the rotation information is not available, e.g., Mocap data, or the application requires both large-scale rotations and translations the non-linear methods are the only choice.

Tables 6.2 and 6.3 provide statistics and runtime results for the examples described above. All runs were performed on a P4 3GHz machine. We use $G(V)$ to measure the difference in shape between the original and deformed models. The value of $G(V)$ for all the models deformed using mean-value encoding is less than $1e^{-3}$ (Tables 6.2 and 6.3). We thus have a numerical indicator that our deformation procedure preserves the local shape of the models. In contrast, when using other methods for the dolphin deformation the error is one or more orders of magnitude larger.

Approach	$G(V)$	Encoding(sec.)	Decoding(sec.)
Mean Value Encoding	0.000232	6.65	0.863
Sheffer and Kraevoy[70]	0.001342	0.455	159551

Table 6.2: Feline deformation statistics (34759 ROI vertices, 9 anchors). $G(V)$ measures the value of the function on the deformed model, given the original mean-value encoding.

Approach	$G(V)$	Encoding(sec.)	Decoding(sec.)
Mean Value Encoding	0.000146	0.190	0.054
Sheffer and Kraevoy[70]	0.001061	0.052	16173
Sorkine et al.[72]	0.006401		
Alexa[4]	0.011042		

Table 6.3: Dolphin deformation statistics (1156 ROI vertices, 1 anchor). $G(V)$ measures the value of the function on the deformed model.

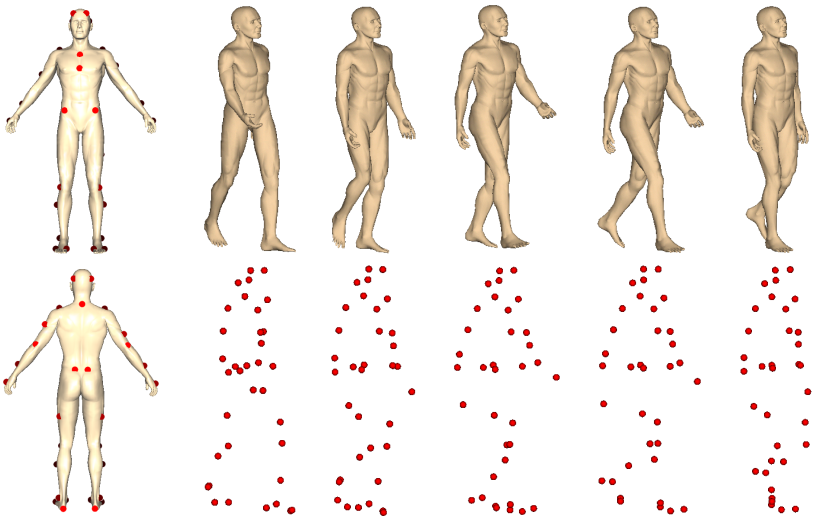


Figure 6.5: Reconstruction of fully realistic complex motion from Mocap data. (a) Marker placement on the surface of the model. (b) (top) Animation sequence. (bottom) Original Mocap data.

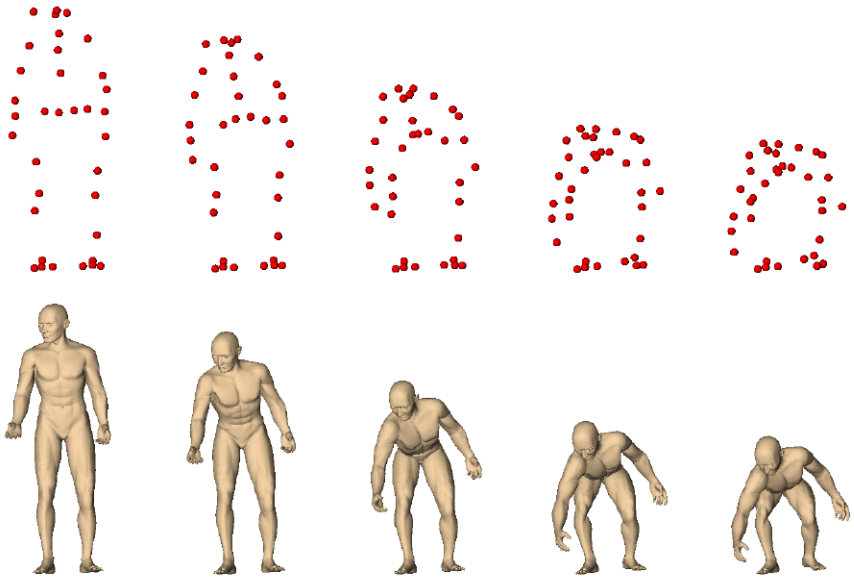


Figure 6.6: Reconstruction of more complex sitting motion from Mocap data. (top) Original Mocap data. (bottom) Animation sequence.

6.2 Conclusion

We introduce a new, robust method for motion reconstruction from Mocap data based on a novel shape representation. Our representation captures the local shape properties of the model and is invariant under rigid transformations, allowing parts of the model to be bent or to rotate during animation. In contrast to standard methods for motion reconstruction, our technique does not require any additional global knowledge of the model structure such as skeleton. Given the input, our technique provides an effective tool for Mocap data reconstruction (Figure 6.5(b)).

Chapter 7

Shuffler: Modeling with Interchangeable Parts

Here we present Shuffler a modeling system for composition from interchangeable parts. Shuffler provides a simple and intuitive composition interface that allows even novice users can create sophisticated models in minutes. Our approach is based on the observation that in many modeling settings users create models which belong to a small set of model classes, such as humans or quadrupeds. The models within each class typically share a common component structure. Following this observation, we introduce a modeling system which utilizes this common component structure allowing users to create new models by shuffling interchangeable components between existing models. To generate new models users simply select which part should come from which input model. As opposed to existing composition tools, Shuffler automatically performs both the boundary specification and the positioning of the parts. For example, the alien in Figure 7.1 was created using just a few part selection choices to combine the body and arm of the standing woman, with the head and feet of the dinopet, the mans legs, and an arm of the sitting woman. The whole process took less than a minute.

Our proposed modeling system combines new algorithms for meaningful mesh segmentation, part matching and intelligent part composition. This dissertation concentrates on the algorithms for meaningful mesh segmentation and intelligent part composition. We consider a perception based definition of meaningful parts [29] and develop a number of quantifiable metrics based on this definition. Our decomposition algorithm uses these metrics to automatically segment 3D models into meaningful parts. We also provide a simple to use, and robust part composition system. Our approach

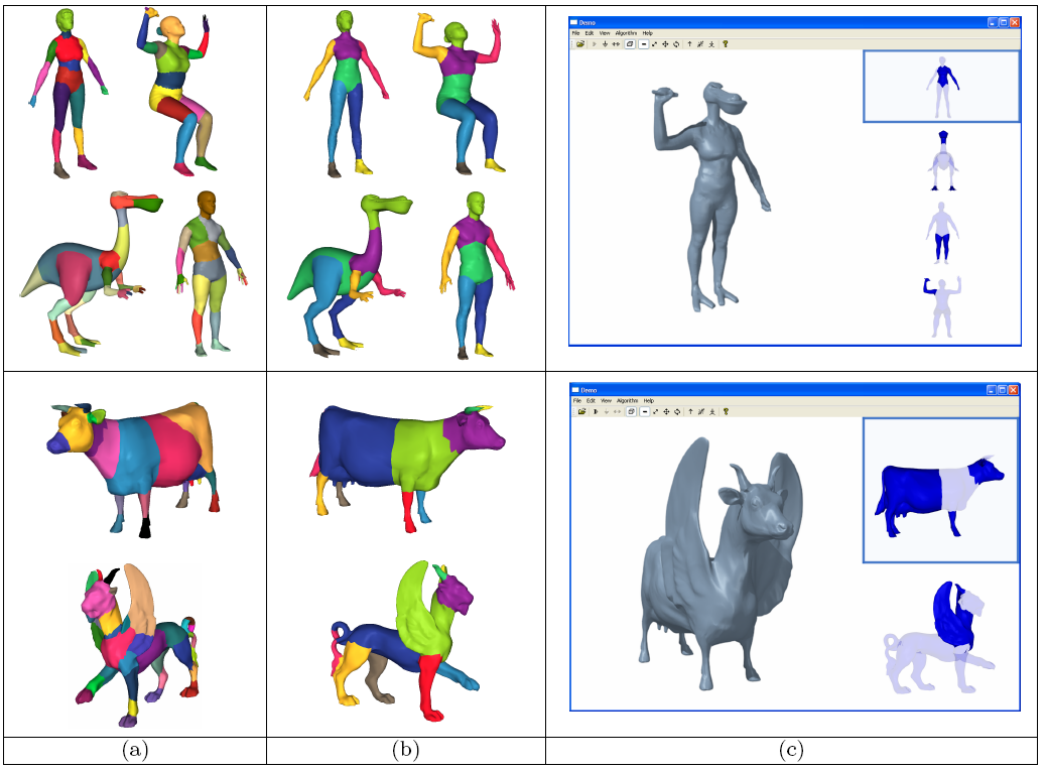


Figure 7.1: Shuffler: (a) Model decompositions computed using our segmentation algorithm. (b) Part correspondences computed using our matching algorithm. (c) An alien and a flying cow generated by shuffling the parts.

significantly simplifies the generation of 3D content, making modeling faster, more accessible, and less expert driven.

7.1 Algorithm

The Shuffler editing system operates on commonly used classes of both natural and man-made objects. It works on both watertight and non-watertight meshes including models with any number of connected components, which are quite common in on-line databases. The system consists of a modeling interface and a pre-processor that segments the input meshes into meaningful interchangeable components. The pre-processing can be performed on the fly once the user selects the set of models they

want to work with or can be carried out beforehand for a database of models.

The accepted notion of meaningful parts relies on human perception, and is based on the observation that human vision defines part boundaries along negative minima of principal curvatures. This definition implies that the meaningful parts are, in some sense, convex.

7.1.1 Convex Segmentation

The goal of our decomposition algorithm is to segment the model into a small number of nearly convex, compact parts. Given a 3D model, the sole input required from the user is a threshold value specifying the convexity of each generated part. In contrast to previous methods we do not expect the user to provide an estimated number of parts for the decomposition.

Metrics: We base our algorithm on a convexity metric that measures the distance between a mesh part P and its convex hull $C(P)$. The distance is defined as an area weighted average of the distances from the part triangles t to the convex hull:

$$dist(P,C(P)) = \frac{\sum_{t\in P} dist(t,C(P)) \cdot area(t)}{\sum_{t\in P} area(t)}, \tag{7.1}$$

where $area(t)$ is the area of the triangle t, and $dist(t,C(P))$ is the distance from the triangle t to the convex hull $C(P)$ along the direction of the triangle's normal. To achieve a useful decomposition, it is not enough for the parts to be nearly convex, they must also be compact. We calculate the compactness as an area to volume ratio of the convex hull C: $comp(C) = area(C)/volume(C)^{2/3}$. We combine the convexity metric with a volumetric measure of compactness to define a cost function for a potential part as

$$cost(P) = (1 + dist(P,C(P))) \cdot (1 + comp(C(P)))^{\alpha}, \tag{7.2}$$

where α controls the trade-off between the two and is a constant for all our examples.

Part Formation: Given these metrics, the goal of our decomposition algorithm is to segment the model into a small number of nearly convex, compact parts such that the convexity error for each part is below the specified threshold D_{max}. To generate the parts we use a modified Lloyd iteration scheme. In contrast to Cohen-Steiner et al. [16]

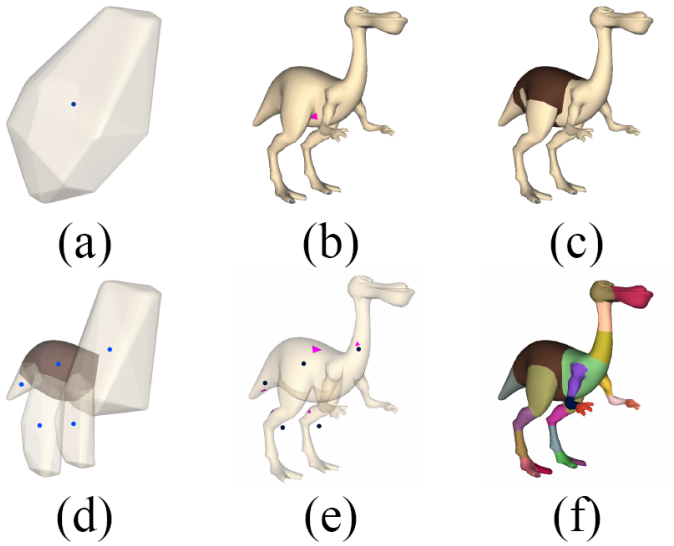


Figure 7.2: Segmentation stages: (a) the hull of the first potential part (the entire model) and its center; (b) first seed triangle; (c) first part after convergence; (d) hulls of the existing part and the new potential parts; (e) new seed triangles; (f) final result.

we do not expect the user to provide an estimate number of parts for the segmentation. Instead we use the threshold to control the number of parts. Starting with zero parts, our algorithm generates the parts incrementally using the following four step procedure.

1. Potential part generation: The method collects the unassigned triangles, which do not belong to any part, into connected components and classifies each component as a potential new part. A triangle can be unassigned either at the beginning of the algorithm or during part growing when all parts have reached the convexity bound D_{max} (Figure 7.2 (c)). Both possibilities are a good indication that a new part is required in the unassigned region. This method of formation of new parts allows us to implicitly handle models with multiple components. After forming the potential parts, the method computes the convex hull for each of them (Figure 7.2 (a) and (d)).

2. Seed generation: This stage finds seeds for re-growing existing and potential parts. A seed consists of a seed triangle and a seed convex hull (Figure7.2 (b) and (e)). We define the seed convex hull as the tetrahedron formed by the seed triangle and the center point of the current convex hull. This selection reduces the difference between

the proxies obtained at consecutive iterations. To further minimize the difference we select a seed triangle which fits well the current proxy, namely one that is close to the current convex hull. After the seeds have been selected, we reset the parts by marking the rest of the triangles as unassigned.

3. Part growing: For each vertex v that shares an edge with a current part, the algorithm computes the insertion cost to be the cost of the updated part formed by adding v to P, $cost(P+v)$. At each growth step, the algorithm uses the cost function to find the best adjacent vertex to add to one of the parts. If the convexity error $dist(P+v, C(P+v))$ of the updated part is below the threshold, the vertex is added to the part. If no such vertex is found growth is terminated and the algorithm proceeds to Stage 4.

4. Termination: The algorithm now tests for termination conditions. If the new parts differ from the ones grown in the previous iteration, the algorithm repeats the reseeding and growing loop, Stages 2 and 3. Once the parts no longer change, we check if the parts cover the entire model. If not, the algorithm returns to Stage 1, otherwise the algorithm terminates.

Figure 7.3 shows a number of decompositions generated using our method. The algorithm correctly detects the perceptually meaningful parts of the models, identifying features like ears and horns, the crown of the triceratops, and the fists of the two women. Our method correctly handles high genus models such as the feline and the fishing reel, which cause problems to many previous methods.

We now establish the correspondences between the parts computed by the segmentation. As noted, since input models may have distinct shapes, their segmentations may differ in terms of the number of parts, the part shape, and the connectivity between the parts. Thus we are unlikely to have a perceptually correct one-to-one correspondence between individual parts. Instead we search to establish a one-to-one correspondence between groups of parts. With the group correspondence defined, the user can start shuffling model parts.

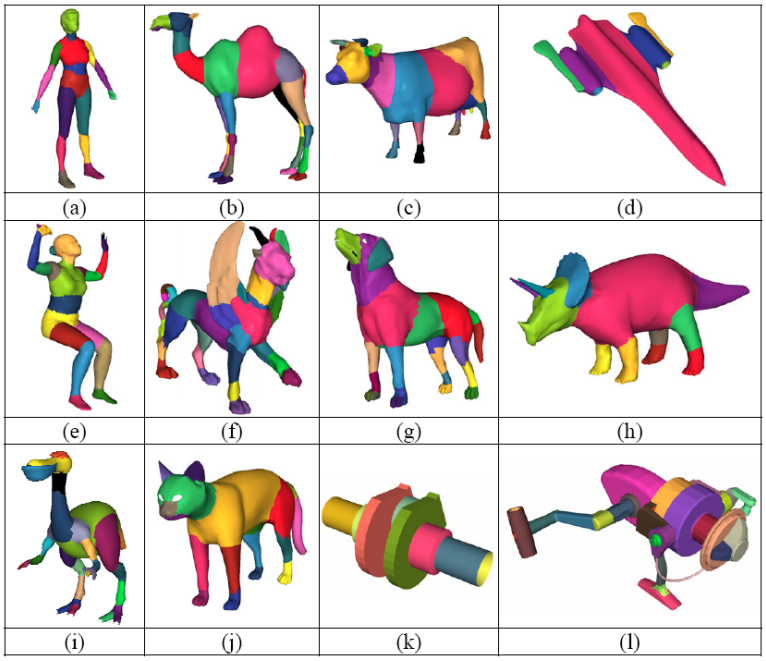


Figure 7.3: Mesh decomposition into meaningful components.

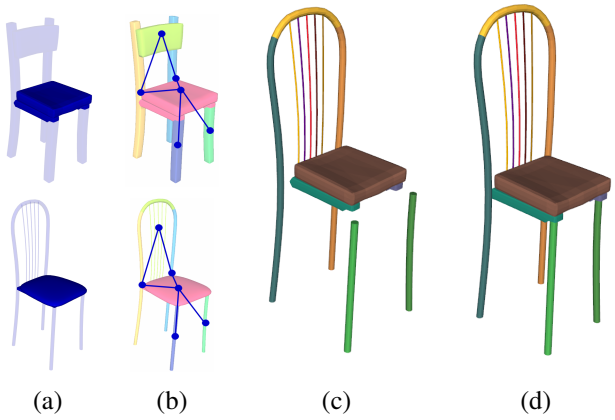


Figure 7.4: Chair seat swap: (a) source (top) and target (bottom) chairs showing the group to be replaced and the replacement in dark blue; (b) a common adjacency graph for both chair models constructed by our algorithm; (c) initial part positioning; (d) positioning of the parts after the automatic alignment.

7.1.2 Intelligent Part Composition

The shuffling interface utilized the component correspondences to provide a mouse-click based composition interface. After a user loads the models of interest, they select a *target* model from which to start the processing. They can subsequently shuffle-in components from the other *source* models by simply clicking on them. As the components are swapped, the system automatically aligns each shuffled-in component with the rest of the target model. If desired, it then blends the composed components together to create a watertight output model.

Alignment: Once the target model is selected, the system automatically aligns the rest of the models with respect to the target model using part correspondence information. The global alignment of source and target models establishes a fixed common frame which can be used as-is to specify the rotation and scale for the shuffled-in component. Alternatively, rotation and scaling can be performed when individual components are shuffled in and out, optimally aligning the convex hulls of the shuffled in and out components using geometric moments. Since the components are nearly convex, the hulls provide a good approximation of their shape. We observe that using the global alignment better preserves the pose of the models typically leading to more visually intuitive results. Shuffler supports both alternatives, using the global alignment as the default.

Shuffler adjusts the translational alignment for each individual shuffling operation to ensure that the constructed model remains connected despite differences in individual component sizes. The alignment is computed using the *common adjacency graph* of the source and target models (Figure 7.4 (b)), which has a node for each pair of corresponding components. The graph has an edge between two nodes if and only if the corresponding components are adjacent on both models. We define two groups to be adjacent if the convex hulls of those groups intersect. We define the midpoint of two groups to be the center of the intersection between their convex hulls. Each edge of the graph is associated with a pair of midpoints. The midpoints serve as the connection points between neighbouring components. We observe that removing the node that corresponds to the shuffled-out component from the adjacency graph can break it

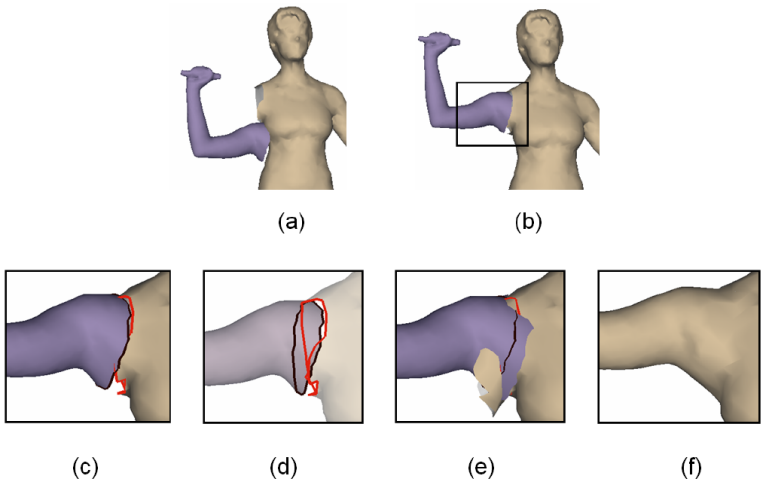


Figure 7.5: Component shuffle: (a) initial position of the shuffled-in component; (b) position after automatic midpoint alignment; (c) and (d) zoom in onto the boundary region; (e) formation of overlap region for blending; (f) final result.

into multiple connected components or branches (Figure 7.4). The algorithm translates each individual branch of the target model to align the shared midpoints between the branch and the shuffled-in component (Figure 7.4 (c) and (d)). In case there is more than one midpoint between the branch and the shuffled-in component it uses the average of the midpoints to calculate the translation. For additional user control we provide an interface to manually adjust the alignment if desired. We observe that using the common adjacency graph Shuffler can compose multiple mesh components in a single operation, something other existing composition algorithms do not support. For models which do not need to be watertight, such as tables or chairs, the process ends here. For watertight models we use a method similar to Sharf et al. [67] to blend the components along shared boundaries. We first extend each boundary by a few layers of triangles (Figure 8) forming an overlap region and then use a variant of soft ICP to snap the two meshes together and generate a common connectivity.

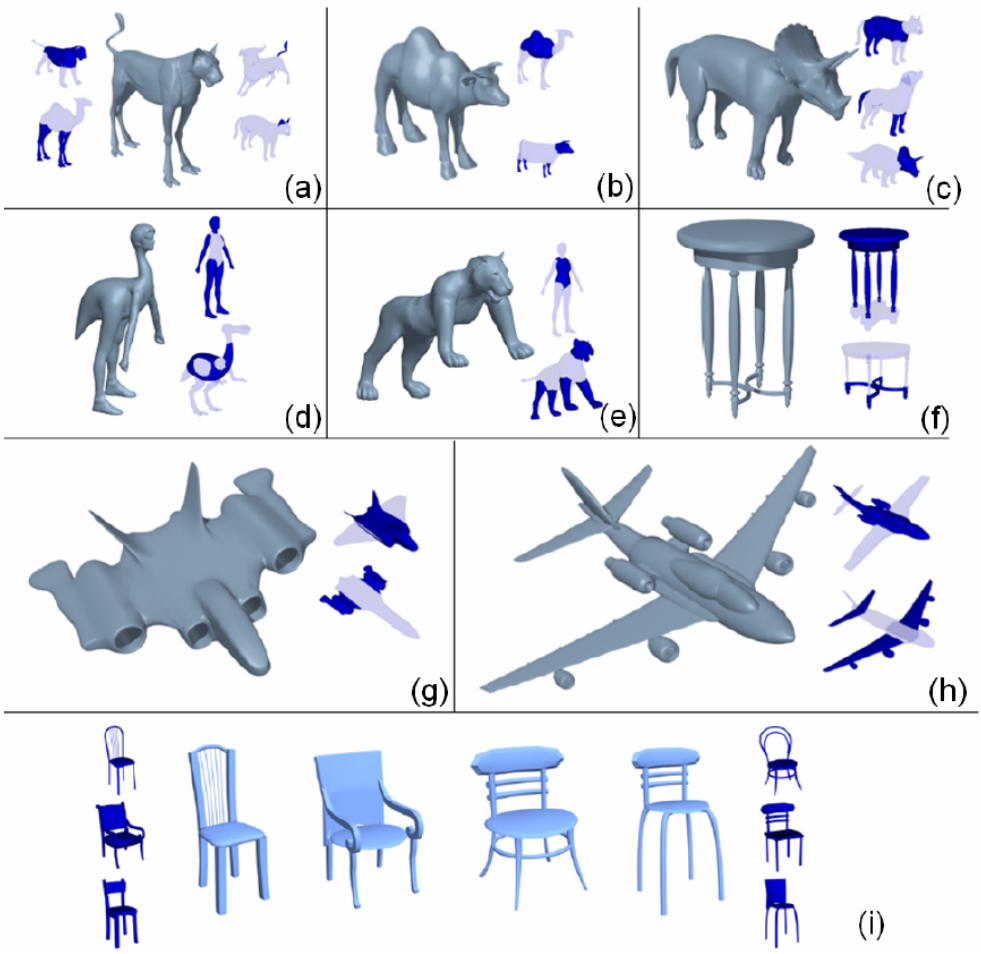


Figure 7.6: Shuffling: (a) cambuliot; (b) camow; (c) triceradog; (d) dinowoman; (e) shelion; (f) a table; (g) a stealth-jet; (h) a six engine super-jumbo; (i) a bunch of chairs.

7.2 Conclusion

We presented a prototype modeling system, Shuffler, which allows users to generate detailed geometric models with a few mouse clicks. As part of the system we developed a method for automatically computing compatible segmentation of models into interchangeable components. We believe such segmentation can be used by many other modeling operations, beyond shuffling. We plan to investigate using the established correspondences to compute cross-parameterization between the components enabling local blending and morphing, as well as transfer of skeletons and associated animations.

Limitations and Future Work

We observe that our definition of parts is purely geometric and does not account for semantics, thus it is not suitable for processing shapes such as faces where some parts (cheeks, forehead, chin) have no clear geometric boundaries in the sense defined by Hoffman and Richards [29]. In our work we did not explicitly consider symmetry, thus the computed segmentations are not necessarily symmetric. Given the recent advances in detecting symmetries in 3D, it would be interesting to introduce symmetry constraints into our system.

Chapter 8

Contour-based Modeling Using Deformable 3D Templates

Another approach to simplify the creation of new models is to provide a simpler interface, for example a sketching tool [31]. Existing sketching tools are good for novices, but practical for creating only simple shapes. We present a new technique for image-based modeling using as input image contours and a deformable 3D template. Our goal is to be able to create new detailed model geometry using information derived from images or drawings. Our work explores how image-based information can be used to produce matching geometry with the help of a 3D template. Figure 8.1 illustrates an example result of our system. The user input consists of tracing the image contours that define the desired proportions and pose of the lion, as well as specifying a small number of point correspondences between the template model and the image. With this input, the output geometry is automatically generated through an iterative match-and-deform process.

Making this type of contour-driven deformation work well requires solving two problems. First, robust correspondences are needed between silhouette vertices on the 3D model and points on the 2D image contours. This is a hard problem because any given silhouette vertex could plausibly match multiple image contour points, or perhaps none, and vice versa. The target contours may also represent a highly deformed version of the template object and thus the silhouette and the contours may differ significantly. Original silhouette vertices may not even be silhouette vertices of the final deformed model. Second, an appropriate deformation model needs to be developed for the 3D template geometry. The template should support a preferred shape, coupled with flex-

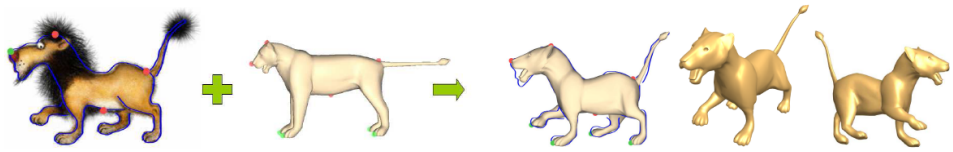


Figure 8.1: Image contours are used to automatically construct a corresponding 3D model with the help of a 3D template. The image contours help inform the pose and proportions of the new shape while the template model helps inform its full 3D shape and surface detail.

ible control of the constraints that will drive the deformation process. It should also support the addition of further knowledge, such as material-like information indicating where the model may prefer to bend, as well as symmetry information.

First, we propose a robust solution to the optimal-correspondence problem by modeling it as a hidden Markov model (HMM). Second, we develop an iterative deformation framework that interleaves finding correspondences with the application of a flexible deformation scheme tailored to our problem domain. The iterative scheme is analogous to iterative closest point (ICP) methods, with the closest point step and the alignment (deformation) step being replaced by methods of appropriate sophistication for the problem at hand. Finally we demonstrate that these techniques can be used to synthesize 3D geometry from 2D images in a way that blurs the distinction between prior work on sketch-based modeling of local deformations and model-based computer vision techniques.

8.1 Algorithm

Our modeling process can be described in terms of a number of steps, as illustrated in Figure 8.2. The first step is the extraction of the input contours, which are created as hand-drawn curves traced over an image or, alternatively, obtained using image processing algorithms. Each contour consists of a sequence of points with associated outward pointing normals. Contour lines are represented as open curves and it is not necessary to provide full coverage of all the silhouette edges in the model.

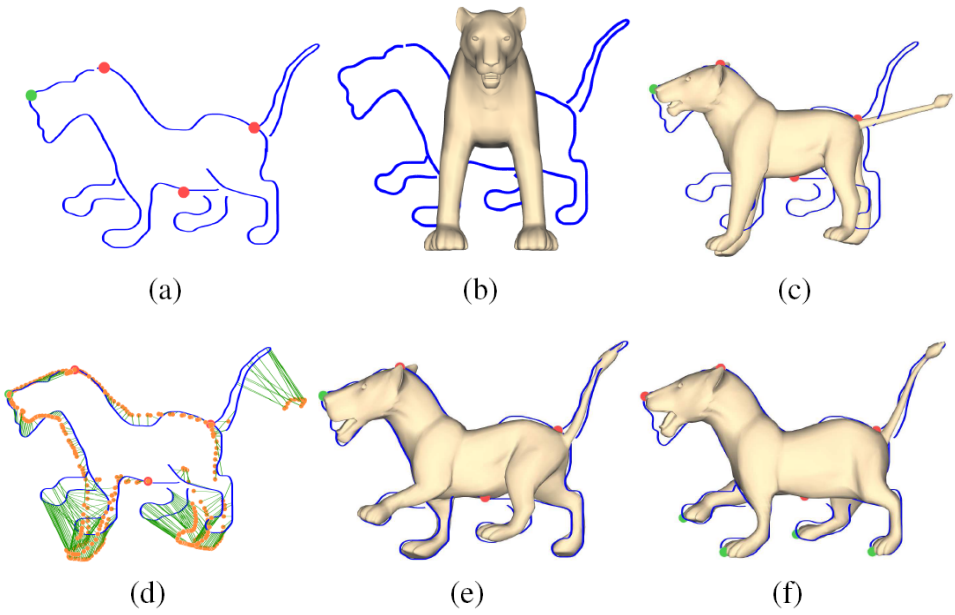


Figure 8.2: Algorithm Overview. (a) Contours obtained from image; (b) Unregistered template model; (c) Initial registration of template model; (d) Initial correspondences; (e) Final deformed model; (f) Final model with extra constraints to enforce the correct mapping of the left and right legs (shown in green).

8.1.1 Initial Registration

Given the input contours and a 3D template, an initial coarse registration needs to be established. This is accomplished using three user-selected contour-to-template (2D point to 3D vertex) correspondences. These points are marked in red in Figure 8.2(a) and the resulting template alignment is shown in Figure 8.2(c). The template is simultaneously rotated and deformed by enforcing the correspondences as hard positional constraints, using the deformation framework to be described in Section 8.1.3. Users can specify additional constraints to resolve ambiguities that arise, such as the left-right reversal that occurs for the solution shown in Figure 8.2(e). The additional constraints can also be used to enforce exact feature correspondences.

8.1.2 Correspondences

After initial registration, we can expect that good correspondences can be established for at least a subset of the contour points and the template vertices. At this point, one could choose to match contour points to template silhouette vertices, or vice-versa, as illustrated in Figure 8.3 (a). We note that both contour points and template model silhouette vertices may not always have a match. Contours may come from an edge-detection algorithm that produces spurious edges, while the template model may contain silhouettes that have no corresponding contour. We choose to look for template silhouette vertices for every contour point. This exploits the strong continuity found in the contour, namely the set of ordered points that comprise the polyline.

Match Criteria: In searching for a mesh silhouette vertex v to correspond to a given contour point p, we wish to optimize proximity and normal difference metrics.

- *Proximity* is measured as $d_P = (p^x - v^x)^2 + (p^y - v^y)^2$ and is optimal when the metric is zero. Note that proximity is measured only in the xy plane, as depth information for the contours is not available.

- *Normal Difference* is measured as the 3D dot product $d_N = n^p \cdot n^v$, where n^p is the normal to the contour at p (lifted to 3D using $z = 0$) and n^v the mesh normal at v. The metric is optimal when the dot product is equal to one.

Considering these two metrics alone, however, neglects the expectation that contours should map to continuous silhouettes. Since each contour is a directed one-dimensional polyline, the points on it can be ordered as $p_1, p_2, ..., p_n$. We found that a simple but effective metric of *Continuity* is the ratio $d_C = \|v_i - v_{i-1}\|^2 / \|p_i - p_{i-1}\|^2$ where v_i and v_{i-1} are the matching vertices of p_i and p_{i-1} respectively, with the distances between the mesh vertices measured in 3D and distances between contour points measured in 2D. The optimal ratio is one, which captures the notion that traveling a given distance along the contour should correspond to traveling a similar distance on the model mesh.

HMM model: To combine the point-to-vertex metric with the continuity metric in a principled way, we cast the problem in terms of a hidden Markov model (HMM). In this framework, contour points are treated as observations and mesh vertices are treated as hidden states, as shown in Figure 8.3(c). The goal is to find the most-likely left-to-right path through the trellis, i.e., the most likely sequence of vertices (hidden states) that could have produced the given contour points (observations). An example solution is illustrated on the trellis, and the induced correspondences are shown in Figure 8.3(d).

The HMM requires *emission probabilities*, i.e., the likelihood that a given hidden state will produce a given output, and *transition probabilities*, i.e., the likelihood of a transition from one hidden state to another. The proximity and normal metrics are used to compute the emission probabilities as follows:

$$P(p_j|v_i) \propto e^{-\frac{1}{2}\left(\frac{d_P}{\sigma_P}\right)^2} e^{-\frac{1}{2}\left(\frac{d_N - 1}{\sigma_N}\right)^2}$$

The continuity metric is used to compute the transition probability:

$$P(v_i|v_{i-1}) \propto e^{-\frac{1}{2}\left(\frac{d_C - 1}{\sigma_C}\right)^2}$$

We use values of $\sigma_P = 0:25D$, $\sigma_N = 1$, and $\sigma_C = 0.05$ for all our examples, where D is the maximum dimension of an input image. The HMM problem is solved using the well-known Viterbi algorithm [63]. User-specified point-to-vertex correspondences force the HMM solution to pass through a given point in the trellis.

The HMM solution may result in several contour points corresponding to the same mesh vertex. A post processing pass remedies this by uniquely assigning them to the

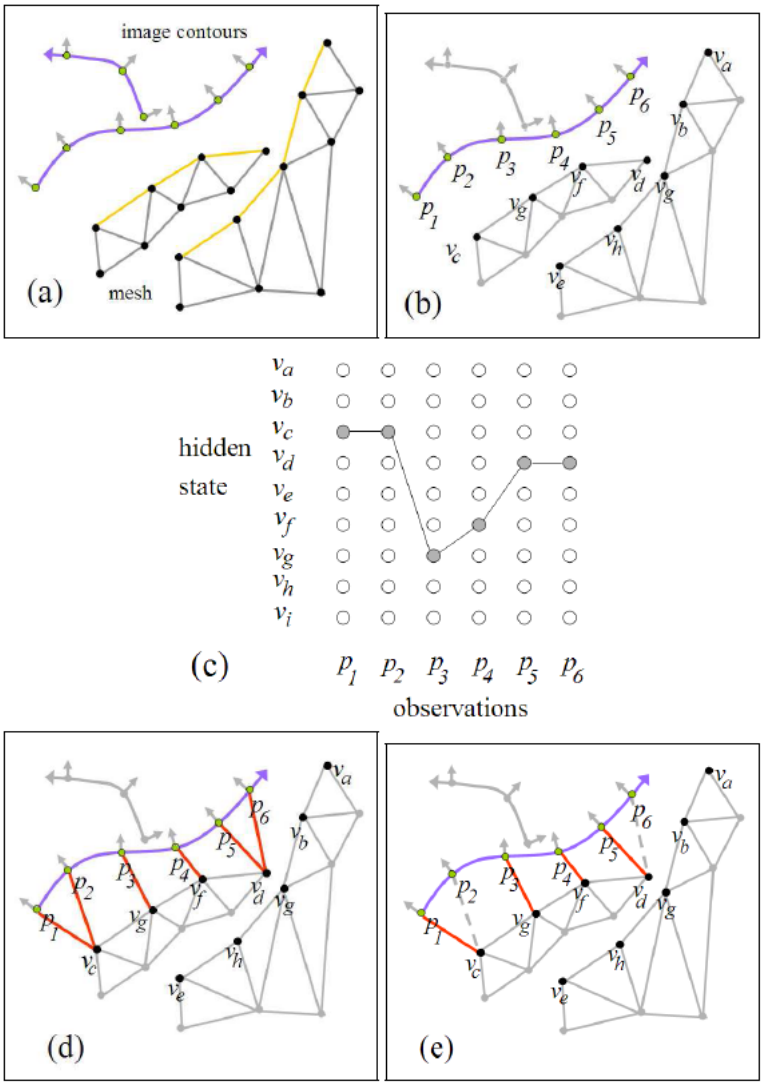


Figure 8.3: Establishment of Correspondences. (a) Input contours and mesh. (b) Each contour point needs to find a best-match vertex. Vertex connectivity is ignored, but taken into account by the transition cost. (c) The problem cast as a hidden Markov model, with the solution path illustrated. (d) Best matches found by the solution path. (e) Elimination of many-to-one matches (dashed lines).

most likely contour point as determined by the proximity and normal difference (Figure 8.3(e)).

8.1.3 Deformation

Given the computed correspondences, we apply a deformation mechanism that attracts the matched vertices to their contour counterparts. To generate the required large deformations, it is necessary not only to pull the vertices towards their corresponding contour points, but to also attract the normals at these vertices towards the corresponding contour point normals. Furthermore, soft rather than hard constraints must be used when attracting the vertices and their normals towards the contours in order to maintain the tradeoff between shape preservation and contour matching, particularly as some of the matches may be inaccurate. The mean-value encoding [44] provides a closed form formulation which can be augmented to support the required types of constraints and therefore it forms the basis of our deformation approach. We note that other non-linear formulations could also potentially be used.

Our algorithm applies an iterative correspond-and-deform approach. At each iteration a set of optimal correspondences is established between the contours and the deforming template model. These are then used in a subsequent deformation step by attracting the model towards the contours. Figure 8.2(d) shows the set of correspondences used for the first iteration of deformation, while Figure 8.2(e) shows the final deformation after a number of correspond-and-deform iterations.

8.2 Experimental Results

Contours do not provide depth or occlusion information, allowing ambiguous interpretation of the described shape. The result of Figure 8.2(e) shows the existence of a left-right ambiguity in that the left legs of the template lion have been matched to the right legs of the image contours, and vice-versa. We introduce several mechanisms that allow the user to introduce additional information into the process for cases where the template shape provides insufficient information to resolve such ambiguities. These

include reconstruction using image contours from multiple viewpoints (Figure 8.4), additional 2D constraints (Figure 8.2(f)), and directly fixing the depth of specific template mesh vertices (Figure 8.6).

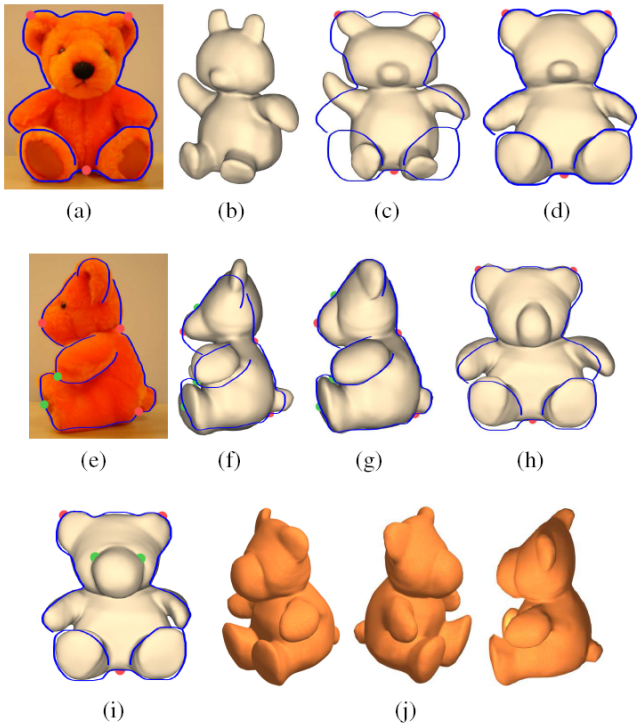
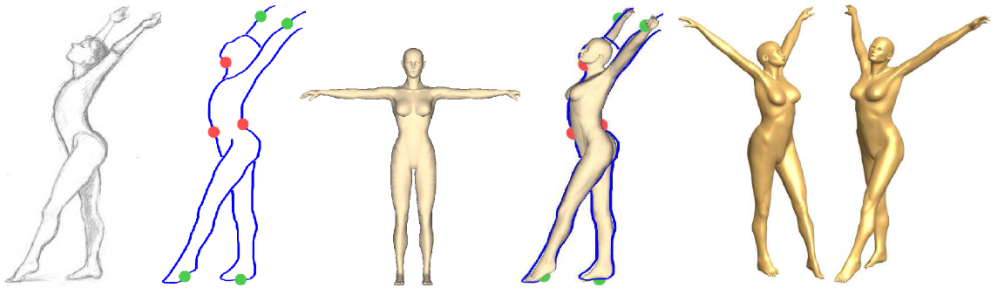


Figure 8.4: Teddy bear: (a) View A; (b) Template model; (c) Initial registration to view A; (d) Final fit to view A; (e) View B; (f) Iniital registration to view B; (g) Final fit to view B; (h) Initial reregistration to view A; (i) Final fit to view A; (j) 3D model.

For multiple views reconstruction we adopt a sequential strategy. In the given example, we deform the template using the contours from view A, then view B, and once again for view A. The image teddy differs from the template teddy in pose as well as the shape of the nose, ears, and feet. These differences are successfully recovered from the contour information.

Achieving correct poses and proportions is a challenging problem for image-based modeling of human figures. To demonstrate that contour based modeling is an effective tool for this task, we apply it to create models from a gymnast drawing and photograph

of a the statue of Hercules. Figure 8.5 shows the results for the female gymnast. Note that the arms are posed using incomplete contours. The correct correspondences are established on the head and result in it facing upwards as in the drawing. Our last example is constructed from photograph of statue. Figure 8.6 shows a muscle-bound Hercules recreated from image contour information. The final model is different from the template in both pose and proportions. In particular, the Hercules statue has large muscle-bound arms and legs as reflected in the final model. The default solution places Hercules right arm slightly in front of the body in a statuesque pose (Figure 8.6(e)). A further constraint placed on the depth value of one hand vertex drives the arm to the correct position behind the body (Figure 8.6(g)).

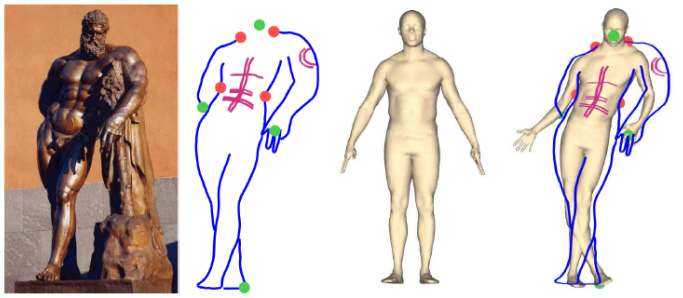


(a) Image (b) Contours (c) Template (d) Final fit (e) 3D model

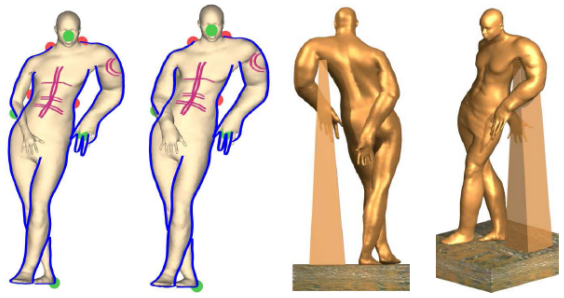
Figure 8.5: Gymnast.

8.3 Conclusions

The algorithms presented provide a powerful framework for developing tailored 3D models from images using only contour information and a flexible process for deforming 3D template models. This is a challenging problem because of the limited and ambiguous nature of the contour information. The HMM-based correspondence model provides the reliable correspondence information that is at the heart of each iteration of the deformation process. The template model itself also includes information that supports reliable model construction, including symmetry information and material



(a) Image (b) Contours (c) Template (d) Registration



(e) Fit, no symmetry (f) Fit with symmetry (g) 3D model, added hand constraints

Figure 8.6: Hercules.

stiffness properties.

Limitations and Future Work

One way of improving our algorithm is to include additional information into the template. For humans and animals, their known skeletal structure could be used to index into a pose-likelihood model, which would help with the process of disambiguating contours or finding sets of likely solutions. Templates could be made to contain information about parts that could be instanced on demand. Templates for more geometric objects could contain information about the precise ways in which their shape can be expected to parameterize.

Chapter 9

Conclusions

In this dissertation, we introduced several efficient and robust 3D data reconstruction and editing tools for model repair and editing pipeline. These tools include techniques for mesh cross-parameterization, model deformation along with model composition and image-based modeling.

We have introduced new methods for computing shape preserving cross-parameter izations and compatible remeshing. Our parameterization method is guaranteed to find a bijective, feature-preserving parameterization for any set of models. The smoothing mechanism we propose significantly reduces the mapping distortion compared to other parameterization methods. Our remeshing method generates compatible meshes for both models with fewer vertices than previous methods while accurately approximating the input geometry. There are multiple applications for the proposed method, including pair-wise model editing [11] transferring texture and surface properties (BRDFs, normal maps, etc) [61], and fitting template meshes to multiple data sets [7, 55].

We extend our cross-parameterization technique to support models with gaps and holes. This allowed us to develop a new and robust method for template-based range scan data completion. Our method is robust in the face of incomplete input meshes with multiple components and holes. Our parameterization mechanism is more generic than previous techniques; it guarantees bijectivity and introduces less distortion. We observe that for models with genus greater than zero, the method may require additional markers, if the resulting patches contain handles. The computed bijective parameterization between the completed model, the template and the base mesh can be used in a variety of applications. An important application of template-based completion is the construction of parameterized shape spaces. Such spaces are useful for both statisti-

cal analysis and the synthesis of new shapes. Our current method requires the user to specify marker vertex correspondences between the input meshes and the template. To parameterize large families of models, such manual selection is impractical. In some cases the markers can be computed from the scan data. In general cases, however, the automation of marker selection requires robust feature-matching techniques. An interesting problem to explore would be an automatic detection of matching features eliminating the need for manual selection.

The creation of novel 3D content is one of the major bottlenecks of modern computer graphics. We proposed various techniques for novel 3D content creation through model deformation or the reuse of existing models, such as model composition or image-based modeling techniques. We introduced a robust method for mesh editing based on a novel local representation, the pyramid coordinates. Our representation captures the local shape properties of the mesh and is invariant under rigid transformations. Using the pyramid coordinates, we developed mesh editing operations which preserve the shape properties of the input models. As a result, our deformation method generates well shaped models even under extremely severe deformations. Our morphing procedure generates intermediate models which interpolate the shape properties of the input meshes. All of the proposed editing operations require minimal user interaction. The method is fast and simple to implement. These properties make pyramid coordinates based editing an effective tool for geometry processing and animation. This dissertation also proposed an extension to pyramid coordinates called mean-value encoding. Like the pyramid coordinates, mean-value encoding is based on a set of angles and lengths describing the position of a vertex with respect to its neighbors. However, in contrast to pyramid, it uses a different local frame definition, leading to a closed form formulation. This enables us to achieve much better results in terms of stability, speed, and shape preservation compared to pyramid coordinates. Here, we also proposed a new application for realistic reconstruction of complex human motion based on Mocap data alone.

Many man-made and natural objects are easily classified into families of models with a similar part-based structure. Example families include quadrupeds, humans, chairs, and airplanes. This dissertation proposes a new method called Shuffler a mod-

eling system that automates the process of creating new models by composing interchangeable parts from different existing models within each family. Our system does not require the users to perform any geometric operations; they simply select which parts should come from which input model, and the system composes the parts together. Our proposed modeling system combines new algorithms for meaningful mesh segmentation, part matching and intelligent part composition. We observe that our definition of parts is purely geometric and does not account for semantics, thus it is not suitable for processing shapes such as faces where some parts (cheeks, forehead, chin) have no clear geometric boundaries in the sense defined by Hoffman and Richards [29]. In our work we did not explicitly consider symmetry, thus the computed segmentations are not necessarily symmetric. Given the recent advances in detecting symmetries in 3D, it would be interesting to introduce symmetry constraints into our system.

Another approach to simplify the creation of new models is to provide a simpler interface. This dissertation also proposed a new technique for image-based modeling that allows a user to easily transform a sketch or picture into a 3D model using a 3D template model. The image contours help inform the pose and proportions of the new shape while the template model helps inform its full 3D shape and surface detail. There are two general directions for improving on this kind of image-based modeling work. One direction looks towards extracting more information from images. Shading or other information could be used to help inform the shape. Additional correspondences could perhaps be automatically identified. It may be possible to combine our work with recent work on object-classification in order to automatically identify the right template model. Another direction looks are including more information into the template. For humans and animals, their known skeletal structure could be used to index into a pose-likelihood model, which would help with the process of disambiguating contours or finding sets of likely solutions. Templates could be made to contain information about parts that could be instanced on demand.

To conclude, this dissertation introduced new methods for mesh cross-parameterization, model deformation along with model composition and image-based modeling. Proposed tools contribute to the model repair and editing pipeline and simplify the task of creating and repairing detailed 3D models.

Bibliography

[1] 3DMax. www.3dmax.com/.

[2] M. Alexa. Recent advances in mesh morphing, 2002.

[3] Marc Alexa. Merging polyhedral shapes with scattered features. *The Visual Computer*, 16(1):26–37, 2000.

[4] Marc Alexa. Local control for mesh morphing. In *Proceedings of the International Conference on Shape Modeling & Applications*, page 209. IEEE Computer Society, 2001.

[5] Alias. Wavefront, www.alias.com/.

[6] Brett Allen, Brian Curless, and Zoran Popović. Articulated body deformation from range scan data. 21(3):612–619, 2002.

[7] Brett Allen, Brian Curless, and Zoran Popovic. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph.*, 22(3):587–594, 2003.

[8] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. *ACM Trans. Graph.*, 24(3):408–416, 2005.

[9] Marco Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation - a comparative study. In *SMI '06: Proceedings of the Shape Modeling International 2006*. IEEE Computer Society, 2006.

[10] Manuele Bicego and Vittorio Murino. Investigating hidden markov models' capabilities in 2d shape classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(2):281–286, 2004.

[11] Henning Biermann, Ioana Martin, Fausto Bernardini, and Denis Zorin. Cut-and-paste editing of multiresolution surfaces. *ACM Trans. Graph.*, 21(3):312–321, 2002.

[12] S. Bischoff and L. Kobbelt. Sub-voxel topology control for level-set surfaces. *Computer Graphics Forum*, 22(3):273–280, September 2003.

[13] Mario Botsch and Leif Kobbelt. An intuitive framework for real-time freeform modeling. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 630–634, New York, NY, USA, 2004. ACM Press.

[14] Mario Botsch, Mark Pauly, Markus Gross, and Leif Kobbelt. Primo: Coupled prisms for intuitive surface modeling. In *Fourth Eurographics Symposium on Geometry Processing*, pages 11–20, June 2006.

[15] Bernard Chazelle, David P. Dobkin, Nadia Shouraboura, and Ayellet Tal. Strategies for polyhedral surface decomposition: An experimental study. In *Symposium on Computational Geometry*, pages 297–305, 1995.

[16] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, 2004.

[17] James Davis, Stephen R. Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. *3dpvt*, 00:428, 2002.

[18] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, New York, NY, USA, 1996. ACM Press.

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

[20] 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.

[21] Akash Garg, Eitan Grinspun, Max Wardetzky, and Denis Zorin. Cubic Shells. In *Symposium on Computer Animation*, 2007.

[22] J. W. Gorman, O. R. Mitchell, and F. P. Kuhl. Partial shape recognition using dynamic programming. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(2):257–266, 1988.

[23] Craig Gotsman, Xianfeng Gu, and Alla Sheffer. Fundamentals of spherical parameterization for 3d meshes. *ACM Trans. Graph.*, 22(3):358–363, 2003.

[24] Eitan Grinspun, Anil Hirani, Mathieu Desbrun, and Peter Schröder. Discrete Shells. In *Symposium on Computer Animation*, 2003.

[25] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. *Computer Graphics Proceedings (SIGGRAPH 99)*, pages 325–334, 1999.

[26] T. Hassner, L. Zelnik-Manor, G. Leifman, and R. Basri. Minimal-cut model composition. In *International Conference on Shape Modeling and Applications (SMI' 05)*, pages 72–81. IEEE Computer Society, June 2005.

[27] Yang He and Amlan Kundu. 2-d shape classification using hidden markov model. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(11):1172–1184, 1991.

[28] Adrian Hilton, Jonathan Starck, and Gordon Collins. From 3d shape capture to animated models. *3dpvt*, 00:246, 2002.

[29] D. D. Hoffman and W. A. Richards. Parts of recognition. *ACM Trans. Graph.*, 18(1–3):65–96, 1984.

[30] Jin Huang, Xiaohan Shi, Xinguo Liu, Kun Zhou, Li-Yi Wei, Shang-Hua Teng, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Subspace gradient domain mesh deformation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1126–1134, New York, NY, USA, 2006. ACM Press.

[31] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[32] CYBERWARE INC. http://www.cyberware.com/.

[33] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 561–566, New York, NY, USA, 2005. ACM Press.

[34] Kolja Kahler, Jorg Haber, Hitoshi Yamauchi, and Hans-Peter Seidel. Head shop: generating animated head models with anatomical structure. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 55–63, New York, NY, USA, 2002. ACM Press.

[35] Takashi Kanai, Hiromasa Suzuki, and Fumihiko Kimura. Metamorphosis of arbitrary triangular meshes. *IEEE Comput. Graph. Appl.*, 20(2):62–75, 2000.

[36] Olga A. Karpenko and John F. Hughes. Smoothsketch: 3d free-form shapes from complex sketches. *ACM Transactions on Graphics*, 25(3):589–598, July 2006.

[37] Sagi Katz, George Leifman, and Ayellet Tal. Segmentation using feature point and core extraction. *The Visual Computer*, 21(8–10):865–875, 2005.

[38] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph.*, 22(3):954–961, 2003.

[39] Youngihn Kho and Michael Garland. Sketching mesh deformations. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 147–154, New York, NY, USA, 2005. ACM Press.

[40] Leif Kobbelt, Jens Vorsatz, and Hans-Peter Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Comput. Geom. Theory Appl.*, 14(1-3):5–24, 1999.

[41] Vladislav Kraevoy, Dan Julius, and Alla Sheffer. Model composition from interchangeable components. *The Visual Computer*, 2007.

[42] Vladislav Kraevoy and Alla Sheffer. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.*, 23(3):861–869, 2004.

[43] Vladislav Kraevoy and Alla Sheffer. Template-based mesh completion. In *Symposium on Geometry Processing*, pages 13–22, 2005.

[44] Vladislav Kraevoy and Alla Sheffer. Mean-value geometry encoding. *International Journal of Shape Modeling*, 12(1):29–46, 2006.

[45] Vladislav Kraevoy, Alla Sheffer, and Craig Gotsman. Matchmaker: constructing constrained texture maps. *ACM Trans. Graph.*, 22(3):326–333, 2003.

[46] A. W. F. Lee, D. Dobkin, W. Sweldens, and P. Schröder. Multiresolution mesh morphing. *Computer Graphics Proceedings (SIGGRAPH 99)*, pages 343–350, 1999.

[47] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and Hans-Peter Seidel. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des.*, 22(5):444–465, 2005.

[48] Bruno Levy. Dual domain extrapolation. *ACM Trans. Graph.*, 22(3):364–369, 2003.

[49] Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition of polyhedra. Technical Report TR06-002, Parasol Lab, Department of Computer Science,Texas A&M University, 2006.

[50] Peter Liepa. Filling holes in meshes. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 200–205, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[51] Second Life. http://secondlife.com/.

[52] Jian Liang Lin, Jung Hong Chuang, Cheng Chung Lin, and Chih Chun Chen. Consistent parametrization by quinary subdivision for remeshing and mesh metamorphosis. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 151–158, New York, NY, USA, 2003. ACM Press.

[53] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.

[54] David G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(5):441–450, 1991.

[55] Stephen R. Marschner, Brian K. Guenter, and Sashi Raghupathy. Modeling and rendering for realistic facial animation. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 231–242, London, UK, 2000. Springer-Verlag.

[56] Takashi Michikawa, Takashi Kanai, Masahiro Fujita, and Hiroaki Chiyokura. Multiresolution interpolation meshes. In *PG '01: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, page 60, Washington, DC, USA, 2001. IEEE Computer Society.

[57] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3):1142–1147, 2005.

[58] M. Pauly, N. J. Mitra, J. Giesen, L. Guibas, and M. Gross. Example-based 3d scan completion. *Symposium on Geometry Processing*, 2005.

[59] Joshua Podolak and Szymon Rusinkiewicz. Robust hole filling using atomic volumes. *Symposium on Geometry Processing*, 2005.

[60] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. *ACM Trans. Graph.*, 22(3):340–349, 2003.

[61] Emil Praun, Wim Sweldens, and Peter Schroder. Consistent mesh parameterizations. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 179–184, New York, NY, USA, 2001. ACM Press.

[62] Long Quan, Ping Tan, Gang Zeng, Lu Yuan, Jingdong Wang, and Sing Bing Kang. Image-based plant modeling. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 599–604, New York, NY, USA, 2006. ACM Press.

[63] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. pages 267–296, 1989.

[64] John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. Inter-surface mapping. *ACM Trans. Graph.*, 23(3):870–877, 2004.

[65] Ariel Shamir. Segmentation and shape extraction of 3d boundary meshes. In *State-of-the-Art Report, Proceedings Eurographics*, 2006.

[66] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. *ACM Trans. Graph.*, 23(3):878–887, 2004.

[67] Andrei Sharf, Marina Blumenkrants, Ariel Shamir, and Daniel Cohen-Or. Snappaste: an interactive technique for easy mesh composition. *Vis. Comput.*, 22(9):835–844, 2006.

[68] A. Sheffer and E. de Sturler. Surface parameterization for meshing by triangulation flattening. *International Meshing Roundtable*, 9:161–172, 2000.

[69] A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications, 2007.

[70] Alla Sheffer and Vladislav Kraevoy. Pyramid coordinates for morphing and deformation. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization,*

and Transmission, 2nd International Symposium on (3DPVT'04), pages 68–75. IEEE Computer Society, 2004.

[71] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[72] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 179–188. Eurographics Association, 2004.

[73] Spore. http://www.spore.com/.

[74] Robert W. Sumner and Jovan Popovic. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3):399–405, 2004.

[75] William Tutte. Convex representation of graphs. *Proc. London Math. Soc., 10*, 1960.

[76] Chen Yang, Dana Sharon, and Michiel van de Panne. Sketch-based modeling of parameterized objects. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 89, New York, NY, USA, 2005. ACM Press.

[77] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. A fast and simple stretch-minimizing mesh parameterization. *smi*, 00:200–208, 2004.

[78] Shin Yoshizawa, Alexander G. Belyaev, and Hans-Peter Seidel. Free-form skeleton-driven mesh deformations. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 247–253, New York, NY, USA, 2003. ACM Press.

[79] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, 2004.

[80] H. Zhang and R. Liu. Mesh segmentation via recursive and visually salient spectral cuts. In *Proceedings of Vision, Modeling and Visualization*, pages 429–436, 2005.

[81] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph.*, 24(3):496–503, 2005.

[82] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 259–268, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.